The Student's Guide to the Secret Art of Debugging

Professor Norm Matloff
UC Davis

September 17, 2001

Why should you use a debugging tool in **ALL** of your programming courses?

(a) To please your professors.

(b) To answer questions on debugging tools on final exams.

(c) Because professional programmers make heavy usage of debugging tools.

(d) To save **yourself** time and frustration.

Answer: Choice (c) is true but the best answer is (d)!

We will use the Data Display Debugger (DDD) debugging tool.

- "You see one, you've seen 'em all."
- DDD is a GUI to the gdb debugger.
- DDD usable on C, C++, Java (DDD version 3.3 onward), perl, etc.

Fundamental Principle of Debugging: Confirmation

Finding your bug is a process of confirming the many things that you believe are true — until you find one which is not true.

Examples of things to confirm:

- Your belief that a variable X = 12 at a certain time.
- Your belief that in a certain if-then-else statement, the "else" gets executed.
- Your belief that in the function call F(Y,5), the parameters Y and 5 are received correctly.

So, how do you confirm such things? Use a debugging tool!

Use (for example) DDD to check each belief, until you find one that is false.

Don't use printf() or cout statements, as they

- make you lose precious time/concentration
- make you do lots of time-consuming editing and recompiling to add/remove them
- are less informative

Our running example: An insert sort.

NumInputs numbers to be placed in array Y, ascending order.

Pseudocode:

```
set Y array to empty
get NumInputs numbers from command line
for I = 1 to NumInputs
   get new element NewY
   find first Y[J] for which NewY < Y[J]
   shift Y[J], Y[J+1], ... to right,
      to make room for NewY
   set Y[J] = NewY
```

Call tree:

```
main() -> GetArgs()
       -> ProcessData() -> Insert() -> ScootOver()
       -> PrintResults()
```

```
// insertion sort, several errors

int X[10],  // input array
    Y[10],  // workspace array
    NumInputs,  // length of input array
    NumY = 0;  // current number of
               // elements in Y

void GetArgs(int AC, char **AV)

{  int I;

   NumInputs = AC - 1;
   for (I = 0; I < NumInputs; I++)
      X[I] = atoi(AV[I+1]);
}

void ScootOver(int JJ)

{  int K;

   for (K = NumY-1; K > JJ; K++)
      Y[K] = Y[K-1];
}

void Insert(int NewY)

{  int J;

   if (NumY = 0)   { // Y empty so far,
                     // easy case
      Y[0] = NewY;
      return;
   }
   // need to insert just before the first Y
   // element that NewY is less than
   for (J = 0; J < NumY; J++)  {
      if (NewY < Y[J])  {
         // shift Y[J], Y[J+1],... rightward
         // before inserting NewY
         ScootOver(J);
         Y[J] = NewY;
         return;
      }
   }
}

void ProcessData()

{

   for (NumY = 0; NumY < NumInputs; NumY++)
      // insert new Y in the proper place
      // among Y[0],...,Y[NumY-1]
      Insert(X[NumY]);
}

void PrintResults()

{  int I;

   for (I = 0; I < NumInputs; I++)
      printf("%d\n",Y[I]);
}

int main(int Argc, char ** Argv)

{

   GetArgs(Argc,Argv);
   ProcessData();
   PrintResults();
}
```

Let's try compiling (note -g option) and running the program:

```
% gcc -o ins -g Ins.c
% ins 12 5 9 3 2 25 8 19 200 10

(no output, no return to OS prompt)
```

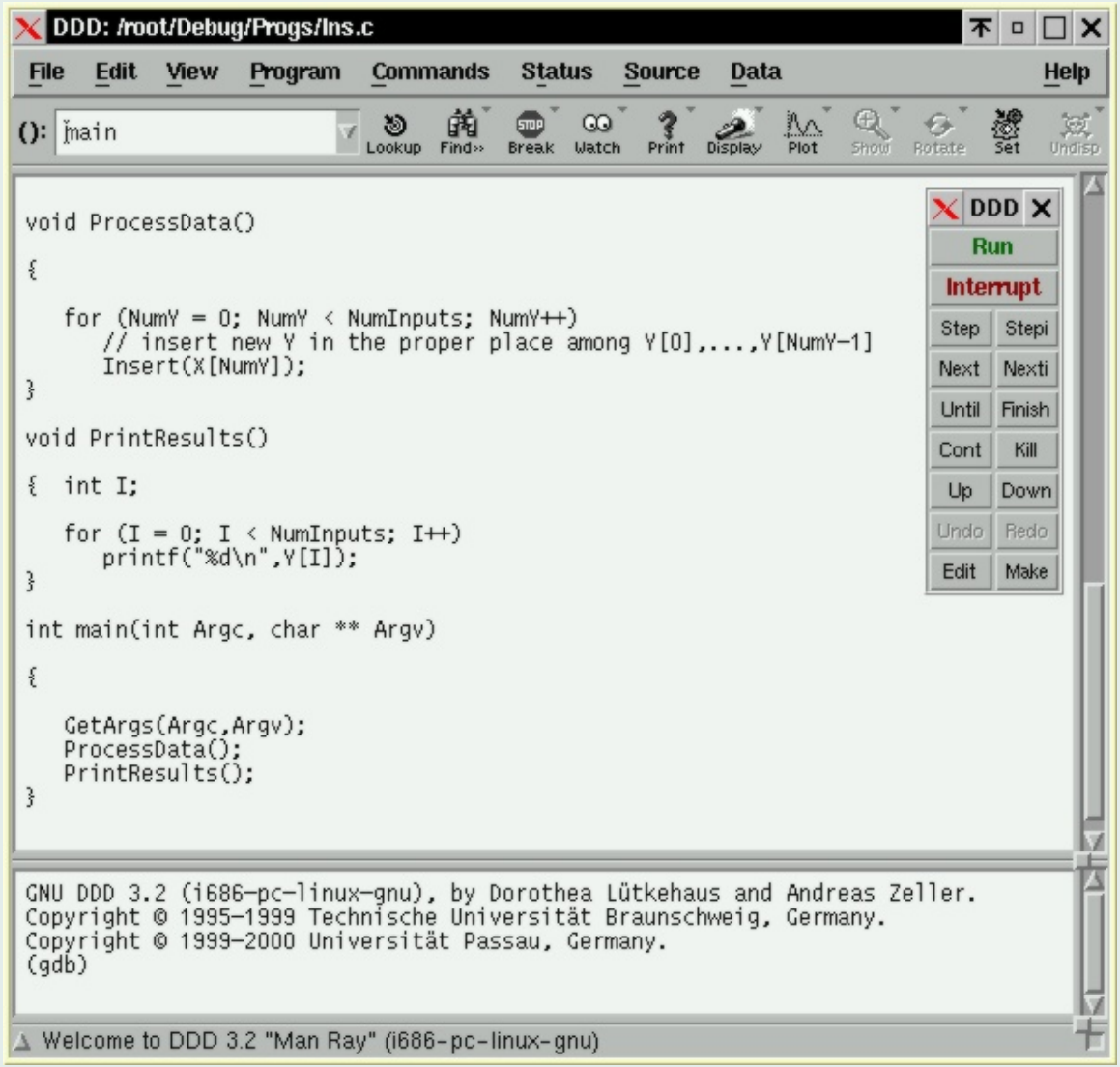
Program just hangs!

OK, let's apply DDD:

```
% ddd ins
```

Or:

```
% ddd --separate ins
```


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): main
Lookup Find» Break Watch Print Display Plot Show Rotate Set Undisp
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the proper place among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
void PrintResults()
{ int I;
for (I = 0; I < NumInputs; I++)
printf("%d\n",Y[I]);
}
int main(int Argc, char ** Argv)
{
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
GNU DDD 3.2 (i686-pc-linux-gnu), by Dorothea Lütkehaus and Andreas Zeller.
Copyright © 1995-1999 Technische Universität Braunschweig, Germany.
Copyright © 1999-2000 Universität Passau, Germany.
(gdb)
Welcome to DDD 3.2 "Man Ray" (i686-pc-linux-gnu)

DDD displays:

- The Source File window, displaying Ins.c.
- The Command Tool window, including buttons such as:
  - Run: Run the program.
  - Step: Execute the current source line, entering called function if any.
  - Next: Same as Step, but do not enter the called function.
  - Cont: Continue execution, not single-stepping.
  - Finish: Execute until we finish the current function.
- The Debugger Console window, showing:
  - GDB commands.
  - Keyboard input and screen output.

The windows will be separate if the –separate option was used when you started DDD.

Well, where should we start?

No magic formula, but a loose guide is:

The Binary Search Principle:

First try to confirm that everything is OK at the "approximate" halfway point in the program. If so, then check at the approximate 3/4-way point; if not, check at the 1/4-way point. Each time, narrow down the big's location to one half of the previous portion.

This of course is not a hard-and-fast, exact rule; it is just a way to suggest where to get started in applying DDD in searching for our bug.

Where is our "approximate halfway point" here in Ins.c? A look at main(),

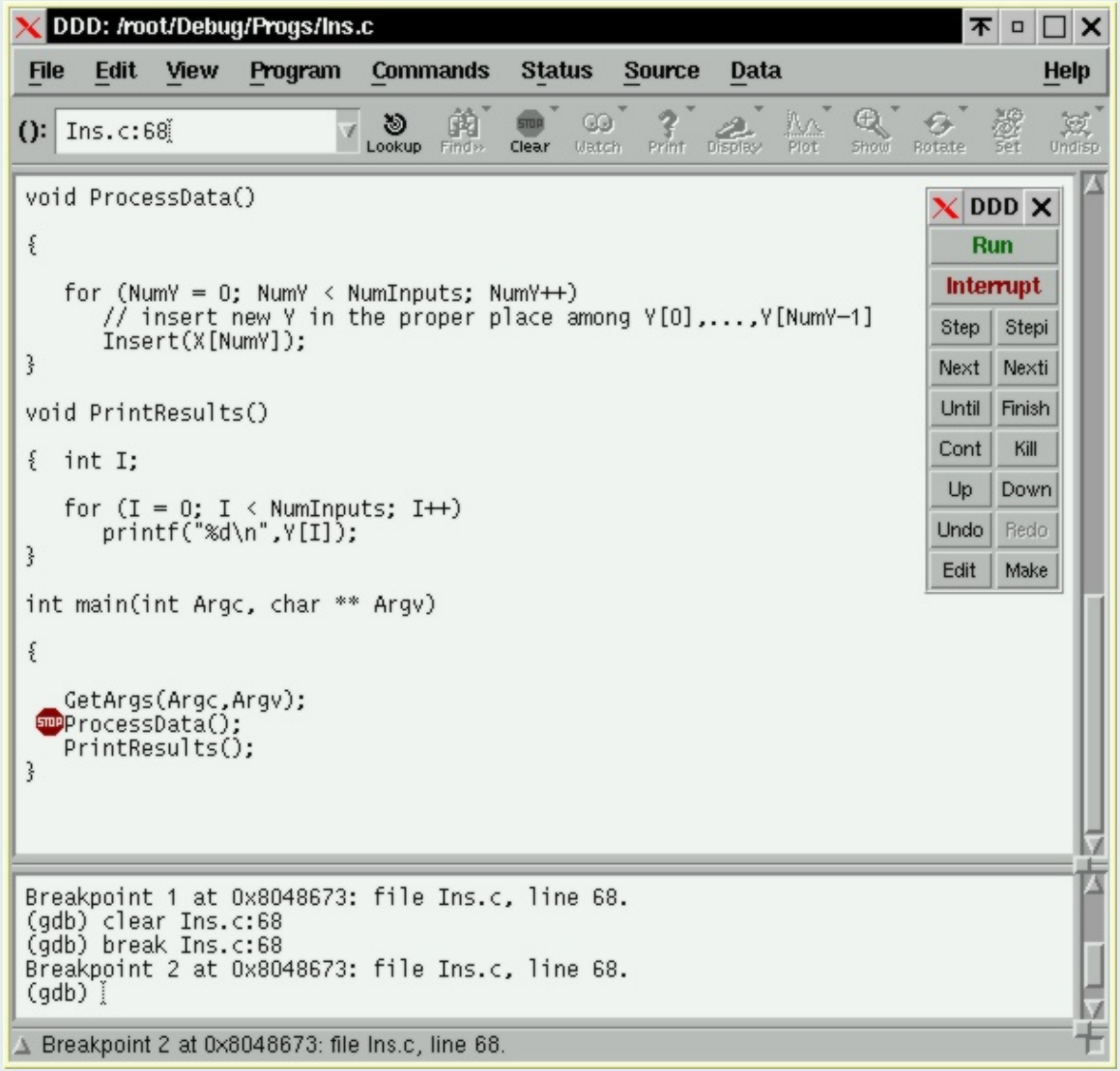
```
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
```

suggests we take as our first confirmation point the beginning of ProcessData().

So, we will set a *breakpoint* — a place where DDD will make execution of the program pause — at the call to ProcessData():

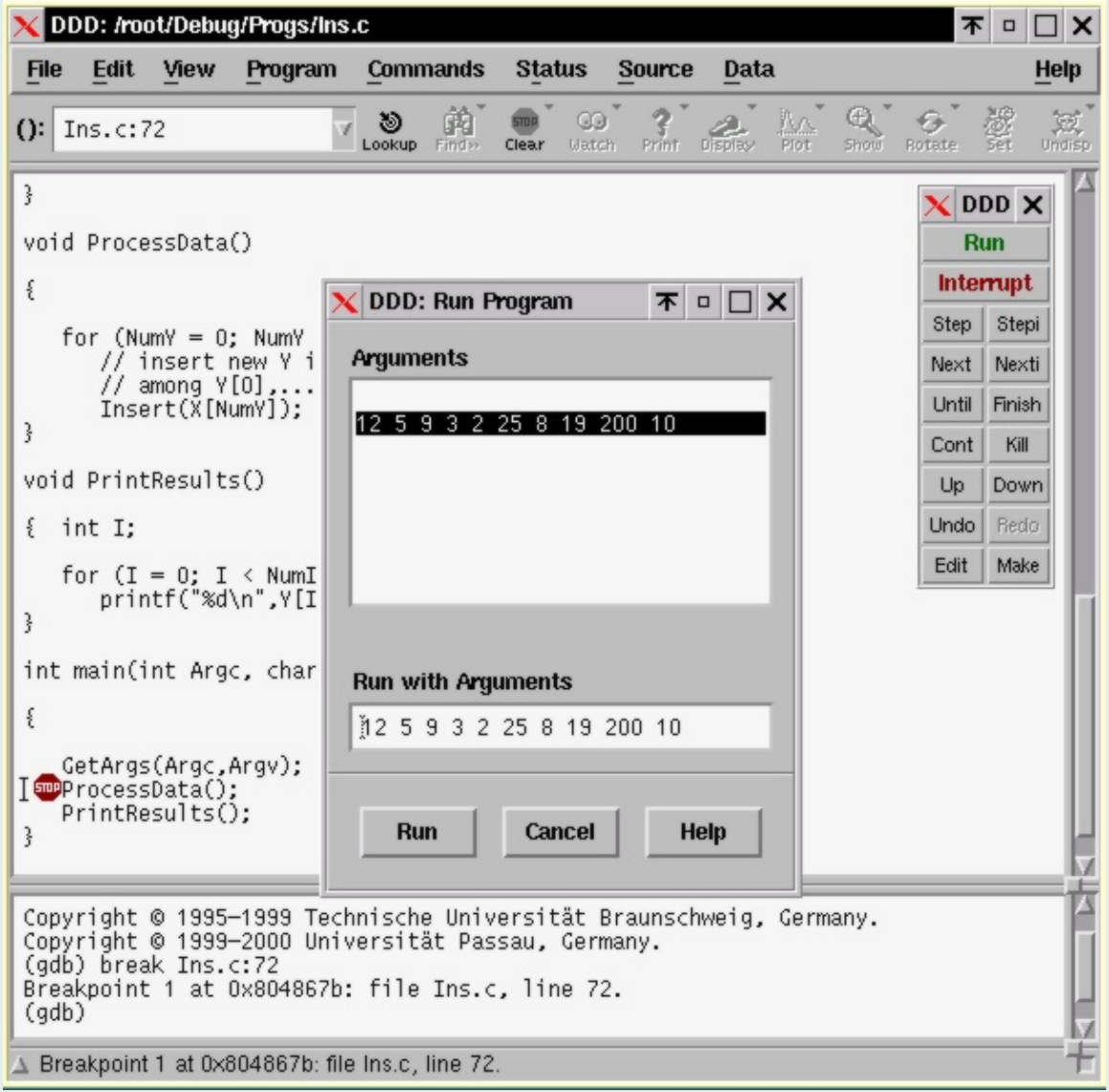
- Move the mouse cursor to the line where ProcessData() is called.
- Click on the left end of the line.
- Click on Break (stop-sign icon).

A red stop sign will then appear on the line, showing that DDD will stop there:

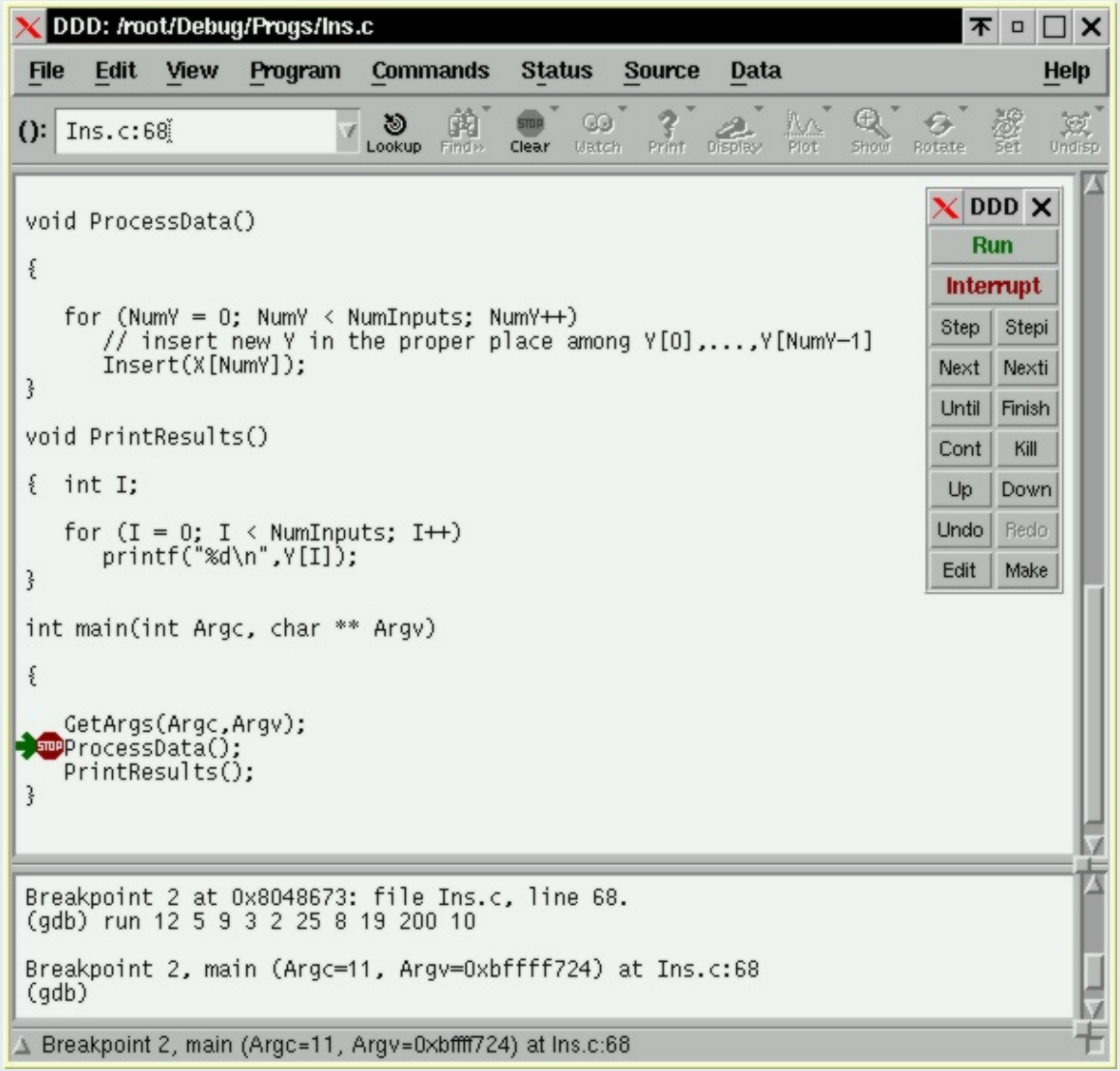

DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Ins.c:68
Lookup Find» Clear Watch Print Display Plot Show Rotate Set Undisp
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the proper place among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
void PrintResults()
{ int I;
for (I = 0; I < NumInputs; I++)
printf("%d\n",Y[I]);
}
int main(int Argc, char ** Argv)
{
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Breakpoint 1 at 0x8048673: file Ins.c, line 68.
(gdb) clear Ins.c:68
(gdb) break Ins.c:68
Breakpoint 2 at 0x8048673: file Ins.c, line 68.
(gdb)
Breakpoint 2 at 0x8048673: file Ins.c, line 68.

Now, let's run the program via DDD:

- Click on Program, then on Run to get the Run Program window.
- Fill in the command line arguments (12, 5, ...) in that window.
- Click on Run in that window.


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Ins.c:72
Lookup Find Clear Watch Print Display Plot Show Rotate Set Undisp
}
void ProcessData()
{
for (NumY = 0; NumY
// insert new Y i
// among Y[0],...
Insert(X[NumY]);
}
void PrintResults()
{ int I;
for (I = 0; I < NumI
printf("%d\n",Y[I
}
int main(int Argc, char
{
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
DDD: Run Program
Arguments
12 5 9 3 2 25 8 19 200 10
Run with Arguments
12 5 9 3 2 25 8 19 200 10
Run Cancel Help
Copyright © 1995-1999 Technische Universität Braunschweig, Germany.
Copyright © 1999-2000 Universität Passau, Germany.
(gdb) break Ins.c:72
Breakpoint 1 at 0x804867b: file Ins.c, line 72.
(gdb)
Breakpoint 1 at 0x804867b: file Ins.c, line 72.

DDD runs our program, stopping at the breakpoint. The green arrow shows our current line, i.e. the one about to be executed:


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Ins.c:68
Lookup Find» Clear Watch Print Display Plot Show Rotate Set Undisp
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the proper place among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
void PrintResults()
{ int I;
for (I = 0; I < NumInputs; I++)
printf("%d\n",Y[I]);
}
int main(int Argc, char ** Argv)
{
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Breakpoint 2 at 0x8048673: file Ins.c, line 68.
(gdb) run 12 5 9 3 2 25 8 19 200 10
Breakpoint 2, main (Argc=11, Argv=0xbffff724) at Ins.c:68
(gdb)
Breakpoint 2, main (Argc=11, Argv=0xbffff724) at Ins.c:68

Now, let's confirm that the program is running correctly so far. All it's done is set NumInputs and the array X, so let's check them:

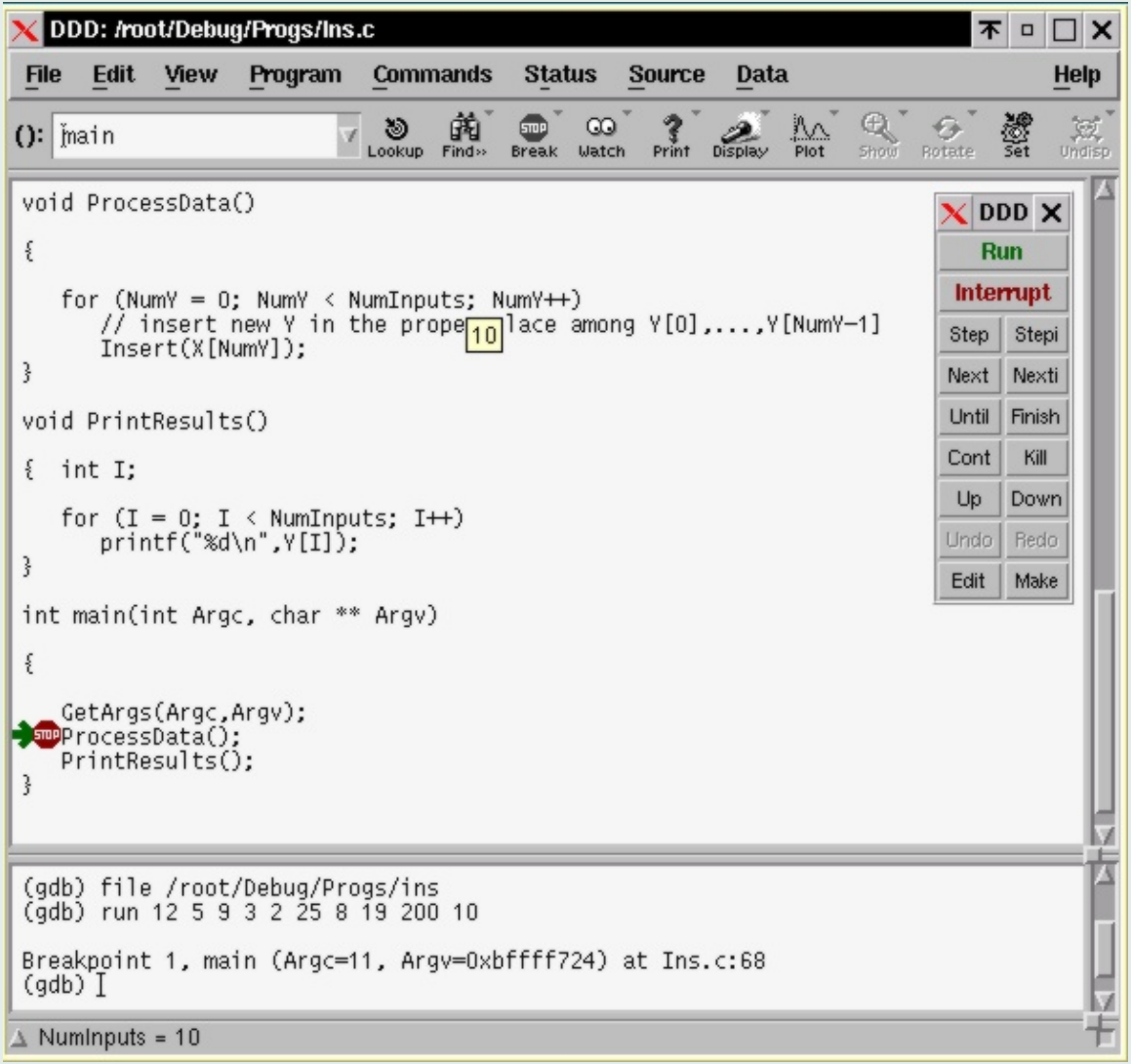
So, let's verify that NumInputs = 10, and that X has the correct values (12, 5, ...):

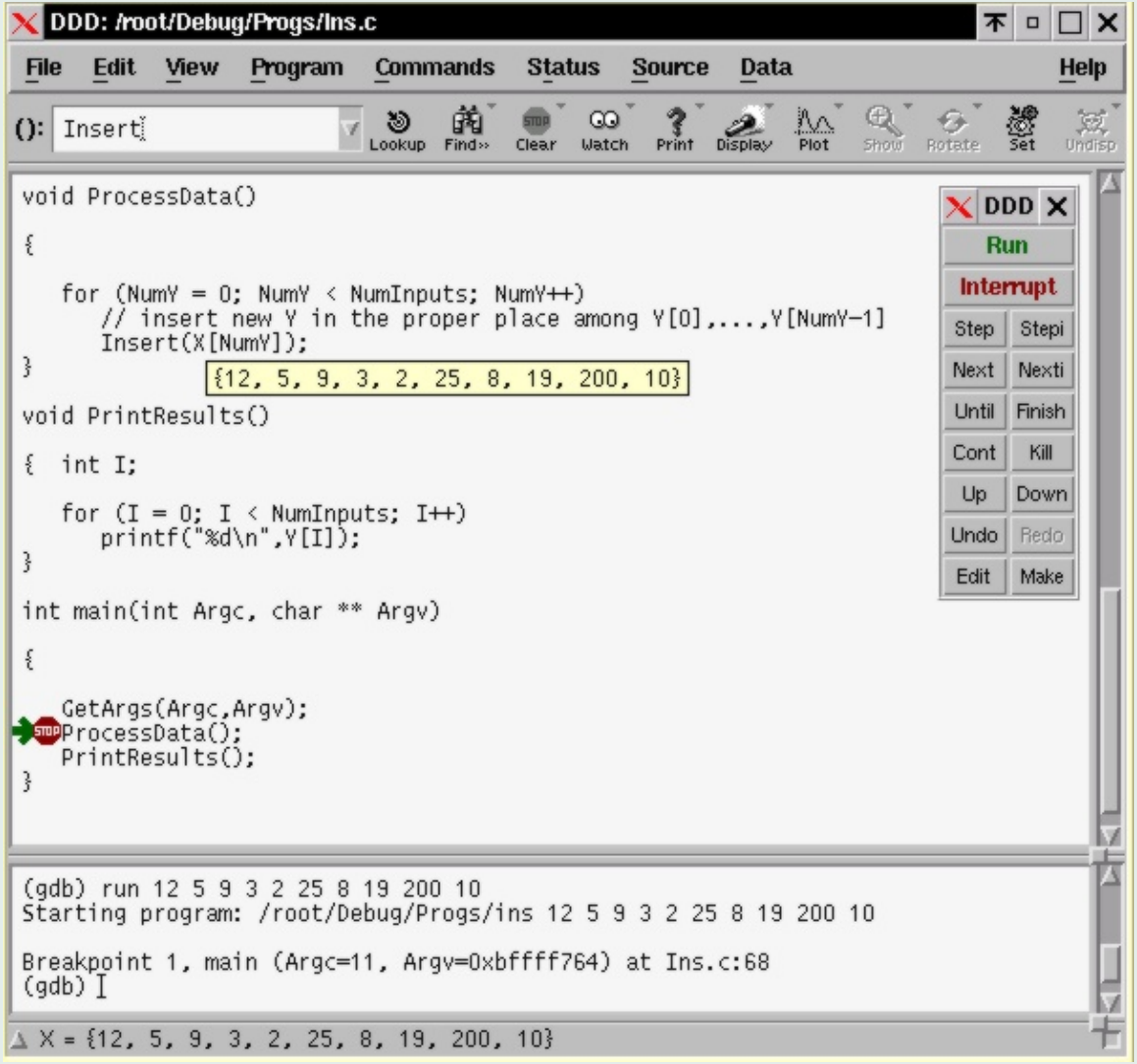
- Move the mouse cursor to any place where NumInputs appears in the Source Code window.

  Rest there for a half second or so.

- The value of NumInputs will appear in a yellow box near the mouse cursor (the cursor doesn't appear here), and at the edge below the Debugger Console.

- Do the same for X.


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): main
Lookup Find» Break Watch Print Display Plot Show Rotate Set Undisp
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the prope 10 lace among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
void PrintResults()
{ int I;
for (I = 0; I < NumInputs; I++)
printf("%d\n",Y[I]);
}
int main(int Argc, char ** Argv)
{
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
(gdb) file /root/Debug/Progs/ins
(gdb) run 12 5 9 3 2 25 8 19 200 10
Breakpoint 1, main (Argc=11, Argv=0xbffff724) at Ins.c:68
(gdb)
NumInputs = 10


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Insert
Lookup Find» Clear Watch Print Display Plot Show Rotate Set Undisp
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the proper place among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
{12, 5, 9, 3, 2, 25, 8, 19, 200, 10}
void PrintResults()
{ int I;
for (I = 0; I < NumInputs; I++)
printf("%d\n",Y[I]);
}
int main(int Argc, char ** Argv)
{
GetArgs(Argc,Argv);
ProcessData();
PrintResults();
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
(gdb) run 12 5 9 3 2 25 8 19 200 10
Starting program: /root/Debug/Progs/ins 12 5 9 3 2 25 8 19 200 10
Breakpoint 1, main (Argc=11, Argv=0xbffff764) at Ins.c:68
(gdb)
X = {12, 5, 9, 3, 2, 25, 8, 19, 200, 10}

OK, both NumInputs and X are all right.

But if there had been a bug within GetArgs(), say

```
X[I] = atoi(AV[I]);
```

we would have found it here, by checking the values of the AV strings, etc.

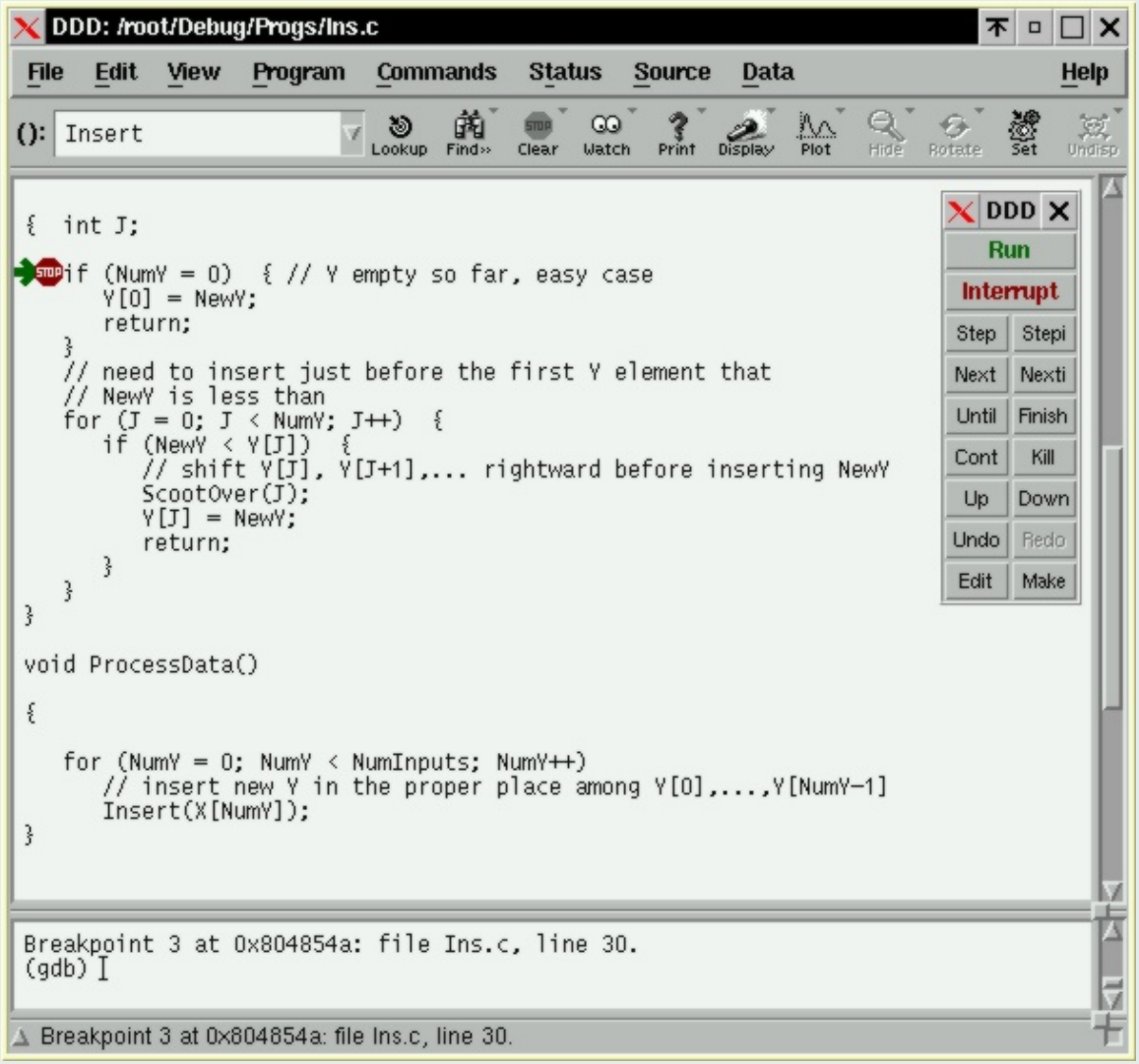
By the way, if we had just wanted to check the value of X[I] here, not all of X (what if X had had 10,000 elements instead of just 10?), we could have used the mouse to highlight the whole expression "X[I]", then right-clicked on it and selected "Print X[I]". The value would be printed to the Debugger Console.

Or we could select "Display X[I]", which would create a sub-window in which X[I] would be continuously displayed.

Let's browse some more, say in Insert().

- Place a breakpoint in the definition of Insert().
- Hit Cont to continue to the next breakpoint, which will be the one we just placed in Insert().
- Once we get into Insert(), the green arrow points to the line

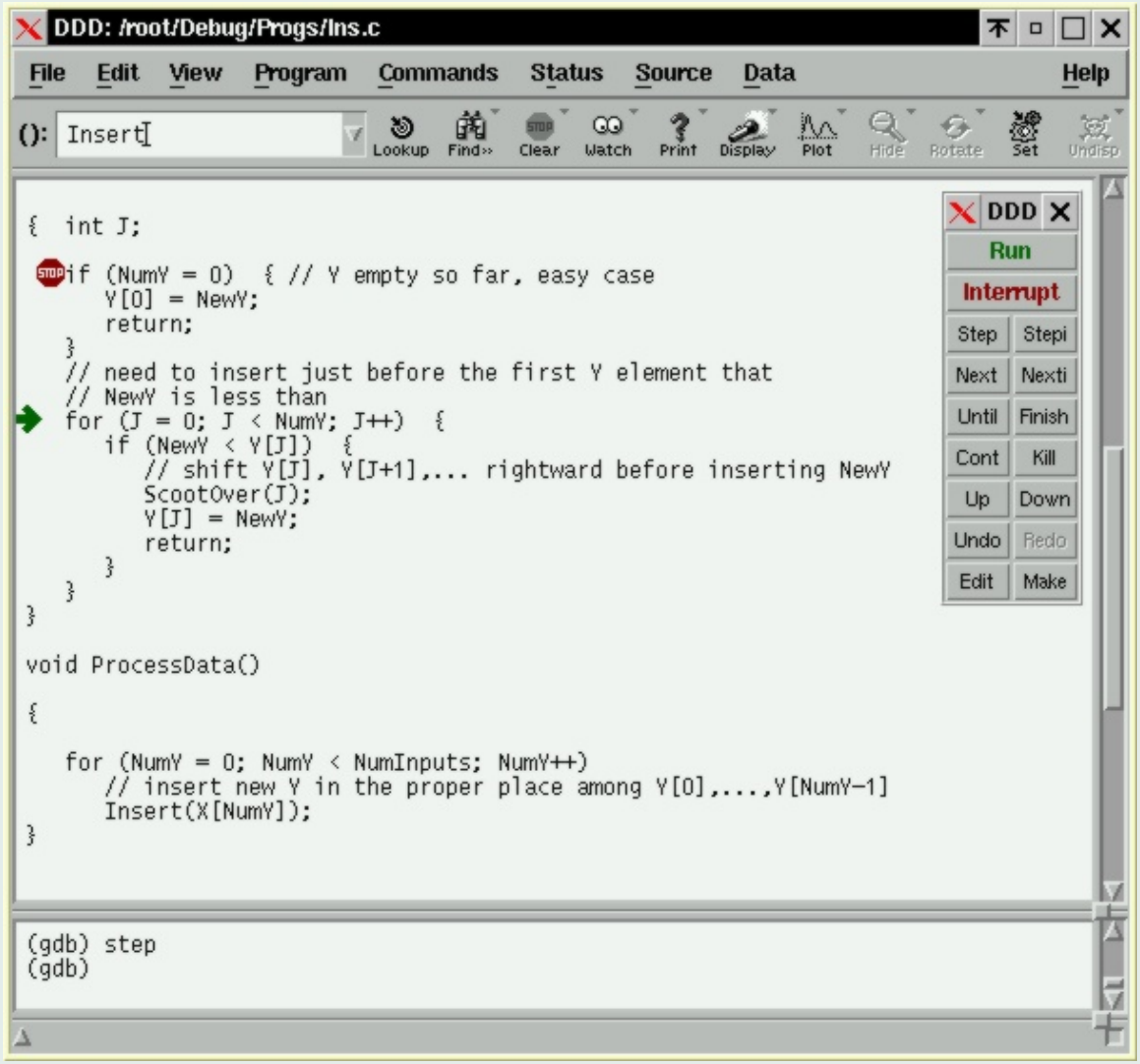
```
if (NumY = 0)  {  // Y empty so far, easy case
```


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Insert
Lookup Find» Clear Watch Print Display Plot Hide Rotate Set Undisp
{ int J;
if (NumY = 0) { // Y empty so far, easy case
Y[0] = NewY;
return;
}
// need to insert just before the first Y element that
// NewY is less than
for (J = 0; J < NumY; J++) {
if (NewY < Y[J]) {
// shift Y[J], Y[J+1],... rightward before inserting NewY
ScootOver(J);
Y[J] = NewY;
return;
}
}
}
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the proper place among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Breakpoint 3 at 0x804854a: file Ins.c, line 30.
(gdb)
Breakpoint 3 at 0x804854a: file Ins.c, line 30.

- This is the first time we've hit Insert(), so NumY should be 0. But we need to confirm it, by moving the mouse cursor to NumY. Yep, it's 0.
- Hitting Step (or Next) that "should" take us to the line

  ```
  Y[0] = NewY;
  ```

  but once again, we must confirm it. Here's what we get:


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Insert
Lookup Find» Clear Watch Print Display Plot Hide Rotate Set Undisp
{ int J;
if (NumY = 0) { // Y empty so far, easy case
Y[0] = NewY;
return;
}
// need to insert just before the first Y element that
// NewY is less than
for (J = 0; J < NumY; J++) {
if (NewY < Y[J]) {
// shift Y[J], Y[J+1],... rightward before inserting NewY
ScootOver(J);
Y[J] = NewY;
return;
}
}
}
void ProcessData()
{
for (NumY = 0; NumY < NumInputs; NumY++)
// insert new Y in the proper place among Y[0],...,Y[NumY-1]
Insert(X[NumY]);
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
(gdb) step
(gdb)

Oh, no! It skipped right over the **if**, going straight to the **for** loop!

So (NumY = 0) must have been false. But we confirmed that NumY was 0!

Hmm...aha! The old classic C learner's error – we used = instead of ==! The **if** line should have been

```
if (NumY == 0)  {
```

That's what made the program hang: It kept assigning 0 to NumY, so the Y array never grew.

We recompile, then run the program outside of DDD, in another window to see if it works now. We get:

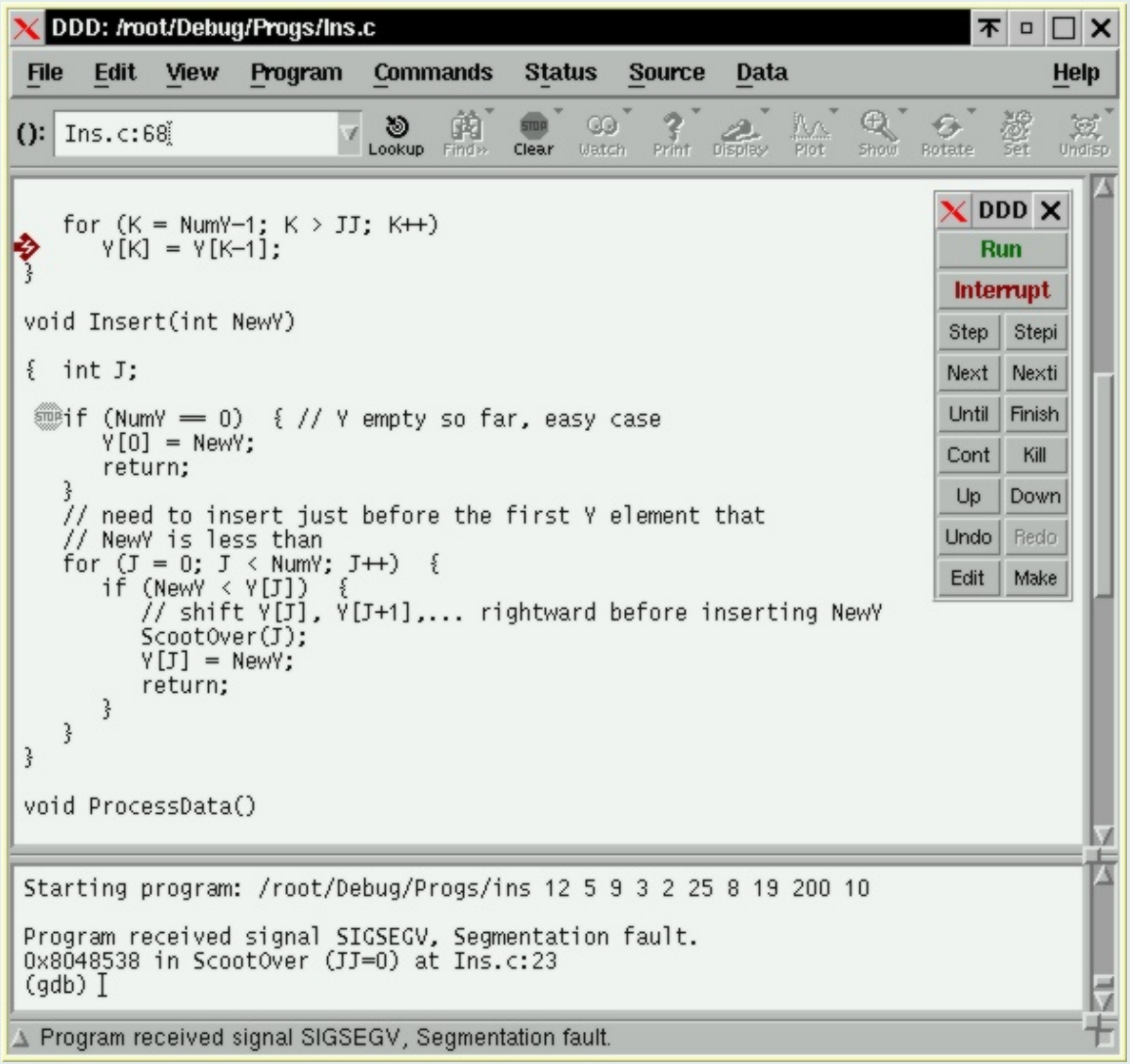
```
% ins 12 5 9 3 2 25 8 19 200 10
Segmentation fault
%
```

A seg fault in Unix means an execution error due to the program's accessing memory which is not allocated to it.

Fundamental Principle for Execution Errors:
The first step to take after an execution error is to run the program through DDD (if the error occurred when run without DDD), to determine where the execution error occurred.

So, let's re-rerun the program within DDD.

- We don't want to stop at the breakpoints, so:
  - Go to each breakpoint.
  - Right-click on the stop sign at the breakpoint.
  - Choose Disable. (The red stop signs turn to gray.)
- We simply hit the Run button in the Command Tool window.
- DDD will automatically load the newly-compiled files.
- We don't have to specify the command-line arguments again, as they are re-used with each run within DDD unless we change them.

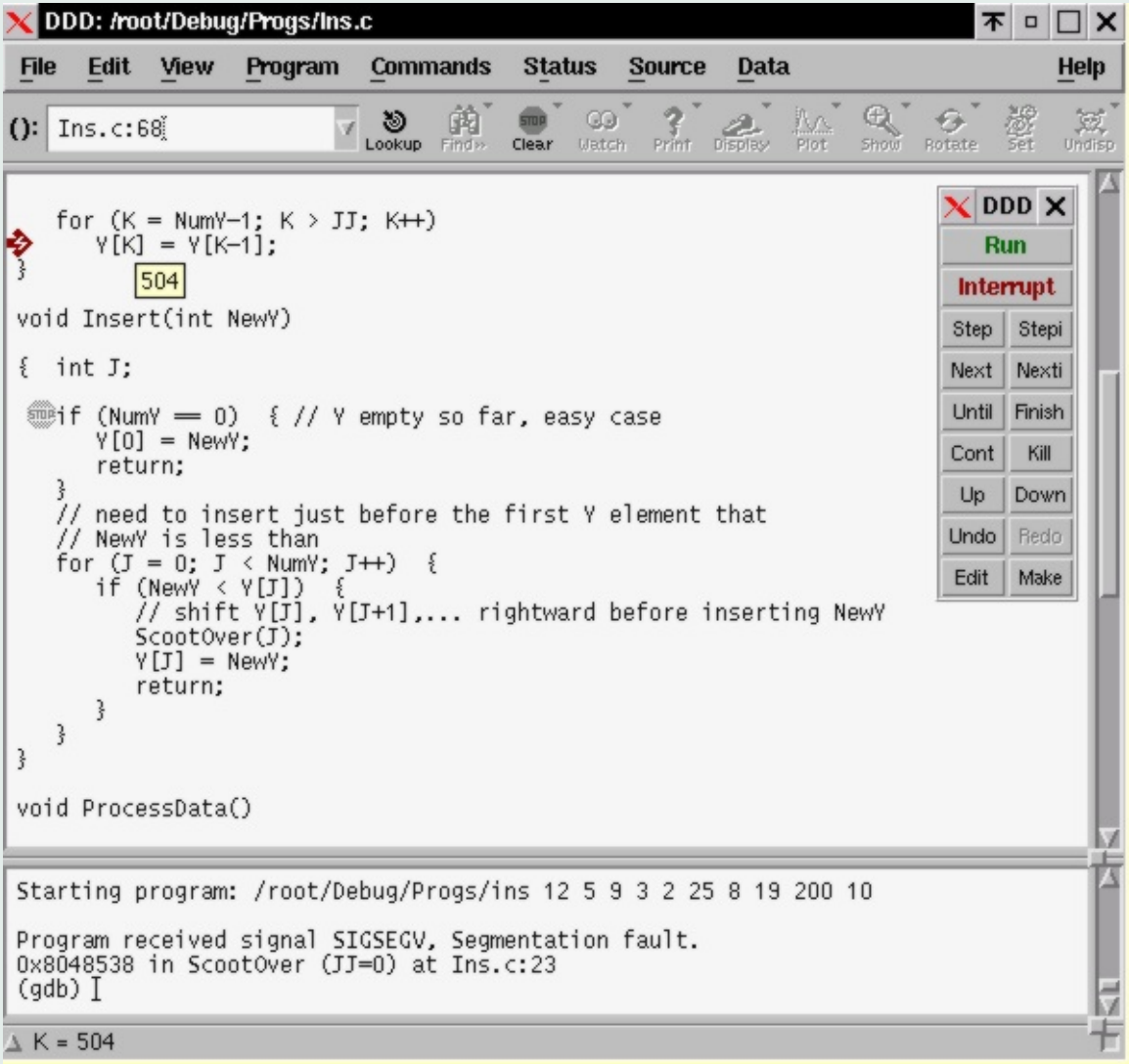

DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Ins.c:68
Lookup Find» Clear Watch Print Display Plot Show Rotate Set Undisp
for (K = NumY-1; K > JJ; K--)
Y[K] = Y[K-1];
}
void Insert(int NewY)
{ int J;
if (NumY == 0) { // Y empty so far, easy case
Y[0] = NewY;
return;
}
// need to insert just before the first Y element that
// NewY is less than
for (J = 0; J < NumY; J++) {
if (NewY < Y[J]) {
// shift Y[J], Y[J+1],... rightward before inserting NewY
ScootOver(J);
Y[J] = NewY;
return;
}
}
}
void ProcessData()
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Starting program: /root/Debug/Progs/ins 12 5 9 3 2 25 8 19 200 10
Program received signal SIGSEGV, Segmentation fault.
0x8048538 in ScootOver (JJ=0) at Ins.c:23
(gdb)
Program received signal SIGSEGV, Segmentation fault.

DDD's red arrow tells us that the execution error occurred on the line

```
Y[K] = Y[K-1];
```

Remember, a seg fault occurs when the program is accessing a portion of memory to which it is not permitted access. So, we suspect that K has a value outside the range of Y, which has only 10 elements.

Let's check, once again by moving the mouse cursor to any place K appears:


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Ins.c:68
Lookup Find Clear Watch Print Display Plot Show Rotate Set Undisp
for (K = NumY-1; K > JJ; K--)
Y[K] = Y[K-1];
}
504
void Insert(int NewY)
{ int J;
if (NumY == 0) { // Y empty so far, easy case
Y[0] = NewY;
return;
}
// need to insert just before the first Y element that
// NewY is less than
for (J = 0; J < NumY; J++) {
if (NewY < Y[J]) {
// shift Y[J], Y[J+1],... rightward before inserting NewY
ScootOver(J);
Y[J] = NewY;
return;
}
}
}
void ProcessData()
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Starting program: /root/Debug/Progs/ins 12 5 9 3 2 25 8 19 200 10
Program received signal SIGSEGV, Segmentation fault.
0x8048538 in ScootOver (JJ=0) at Ins.c:23
(gdb)
K = 504

Whoa! K = 504 is way out of range.

(By the way, K = 10 or 11 did not cause a seg fault, as we were still in the same page of virtual memory.)

Let's look at the code which is setting K:

```
for (K = NumY-1; K > JJ; K++)
   Y[K] = Y[K-1];
```

Recall that this code was supposed to shift the Ys over to the right, first moving the rightmost Y, then shifting the next-to-rightmost Y, etc.

In other words, this was supposed to be a "down" loop; K++ should be K–.

OK, we recompile and then re-run:

```
% ins 12 5 9 3 2 25 8 19 200 10
2
3
5
0
0
0
0
0
0
0
%
```

Not correct yet, but at least the first 3 elements are correct.

Let's go to ProcessData() and do some spot checks for various values of NumY in the loop there:

```
for (NumY = 0; NumY < NumInputs; NumY++)
   // insert new Y in the proper place
   // among Y[0],...,Y[NumY-1]
   Insert(X[NumY]);
```

Trying the Binary Search Principle, let's first do a check for the NumY value which is halfway through, i.e. NumInputs/2 = 5:

- Put a breakpoint on the line which calls Insert().
- Right-click on this breakpoint's stop-sign icon.
- Select Properties.
- Specify Condition as NumY == 5 (not NumY = 5!).
- Hit the Run button.

DDD will stop at the call to Insert() when NumY is 5, skipping the previous calls to Insert().

This is a small convenience for this case in which X is only 10 elements, but would be crucial of X had 10,000 elements.

We then check Y (by moving mouse cursor to Y). It turns out that

```
Y = {2,3,5,0,0,0,0,0,0,0}
```

just as after the end of full execution.

So, the problem occurred during "the first half." Well, let's start at the beginning, since the first X, 12, doesn't appear in Y:

- Remove the Condition NumY == 5 from the breakpoint at the call to Insert(): Right-click on its stop sign icon, choose Properties, erase the Condition.
- Hit the Run button to restart.
- Each time DDD stops at the call to Insert(), do:
  - Check Y (as usual, by moving the mouse to an instance of Y anywhere in the code) to confirm it is what it should be at this stage.
  - Hit the Cont button to have DDD continue until the next breakpoint (which will be this one again).

We find that the case NumY = 0 works fine, i.e. Y[0] = 12. But the next case, NumY = 1, fails: Y[0] becomes 5, and the 12 disappears.

So, we've narrowed the problem down to the events within Insert() when NumY = 1.

Here's the main part of Insert():

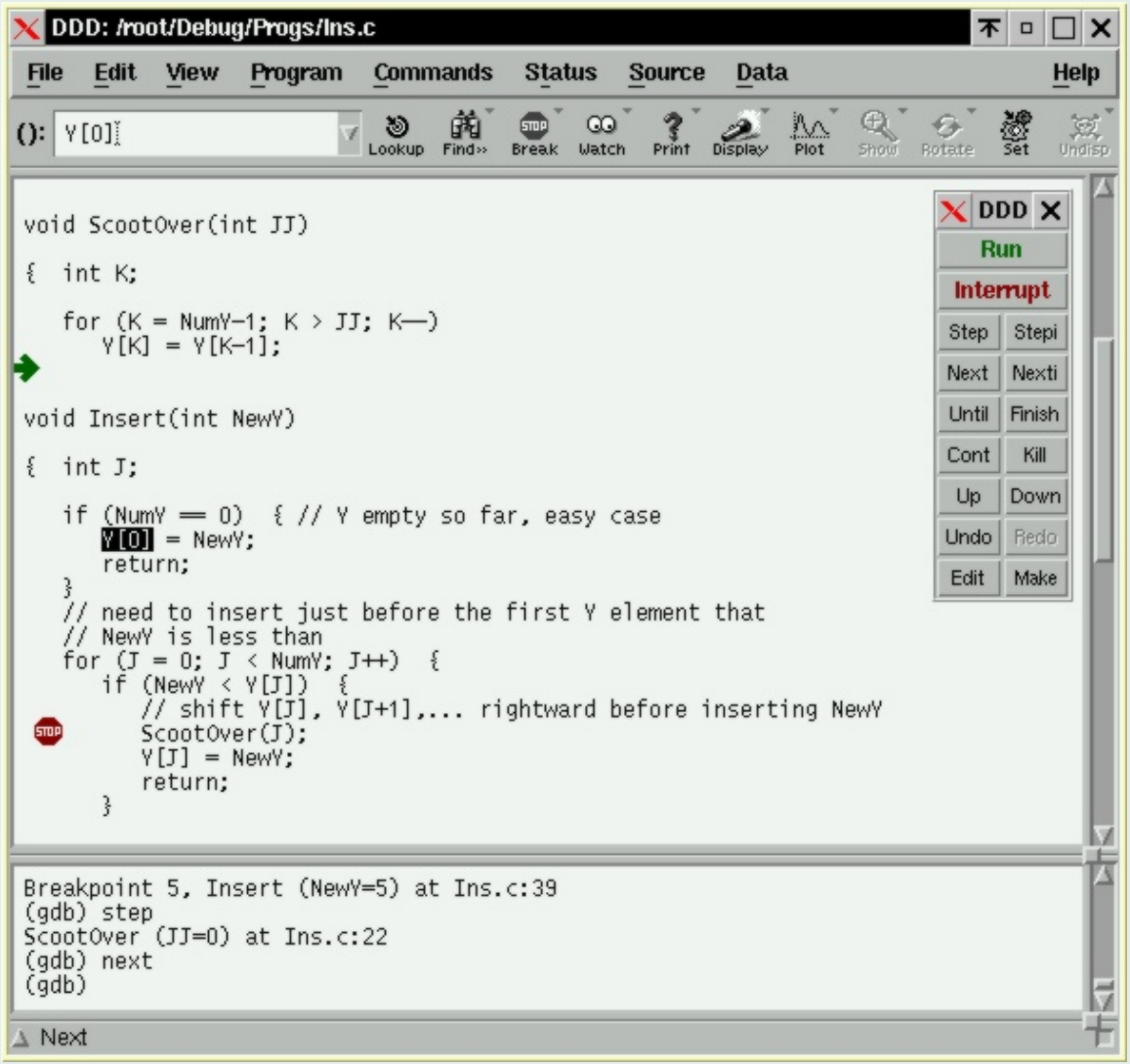
```
for (J = 0; J < NumY; J++)  {
   if (NewY < Y[J])  {
      // shift Y[J], Y[J+1],... rightward
      // before inserting NewY
      ScootOver(J);
      Y[J] = NewY;
      return;
   }
}
```

We need to step through these lines for this case NumY = 1:

- Re-run by hitting Run.
- The first time we hit the breakpoint at the call to Insert(), NumY is 0, so hit Cont.
- Hit Step (not Next) to enter Insert().
- Hit Step or Next twice to get to the **if** within the **for** loop.
- Take a look around: Confirm our expected values for NumY (should be 1), NewY (should be 5), and Y (should be 12, 0, 0, ...). Yes, all OK.
- Now `(NewY < Y[J])` should be true; confirm it: Hit Step or Next. Yes, we do go to the call to ScootOver().
- Let's skip over ScootOver(), i.e. execute it but not enter it: Hit Next (not Step).
- Let's confirm that the Y array was shifted rightward: Move the mouse cursor to display the values in Y. Aha! The 12 is still in Y[0], instead of in Y[1].

So, we need to go into ScootOver().

- Disable existing breakpoints, and add a new one at the call to ScootOver().
- Re-run the program.
- The first time the breakpoint is hit, this should be the case NumY = 1, but confirm it. Yes, OK.
- Hit Step (not Next) to enter ScootOver().
- Remember, at this time Y consists of just one element, 12, which we must shift to the right to make room for NewY, 5, which is to be inserted. We will try to confirm this.
- Hit Step or Next to execute the first line.


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Y[0]
Lookup Find» Break Watch Print Display Plot Show Rotate Set Undisp
void ScootOver(int JJ)
{ int K;
for (K = NumY-1; K > JJ; K--)
Y[K] = Y[K-1];
void Insert(int NewY)
{ int J;
if (NumY == 0) { // Y empty so far, easy case
Y[0] = NewY;
return;
}
// need to insert just before the first Y element that
// NewY is less than
for (J = 0; J < NumY; J++) {
if (NewY < Y[J]) {
// shift Y[J], Y[J+1],... rightward before inserting NewY
ScootOver(J);
Y[J] = NewY;
return;
}
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Breakpoint 5, Insert (NewY=5) at Ins.c:39
(gdb) step
ScootOver (JJ=0) at Ins.c:22
(gdb) next
(gdb)
Next

The loop to shift Y rightward wasn't executed at all.

A closer look shows why: Since NumY was 1 and JJ was 0, the loop amounted to

```
for (K = 0; K > 0; K--)
```

so the loop wasn't executed.

A bit more thought reveals that the expression NumY-1 in the loop should have been NumY.

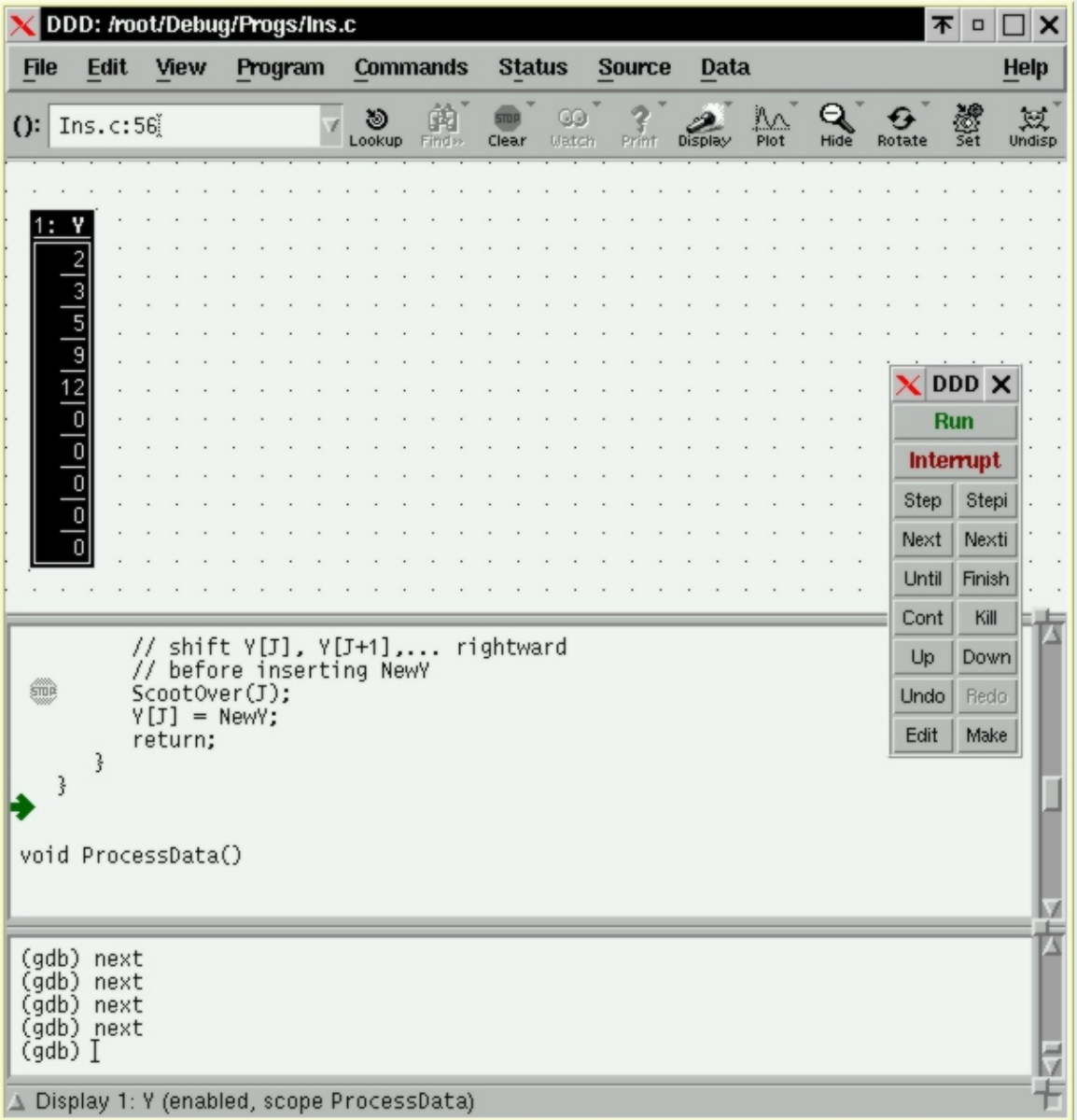
So, recompile and re-run:

```
% ins 12 5 9 3 2 25 8 19 200 10
2
3
5
8
9
10
12
0
0
0
```

Looks like we are almost done, possibly with just one bug remaining.

The first element missing in the output above is 25, so let's see what happened when it was inserted into Y.

- Re-enable the breakpoint at the call to Insert().
- Add the condition X[NumY] == 25.
- Hit the Run button, then Step to enter Insert().
- We anticipate that we may be checking Y a lot, so it's easier to continuously display it:
  - Go to any instance of Y in the Source File window.
  - Right-click on it.
  - Select "Display Y".
  - A Data window will open above the Source File window.
  - You may wish to resize the DDD window, and the Data and Source File windows.
- Keep hitting Next until we see a change in Y.


DDD: /root/Debug/Progs/Ins.c
File Edit View Program Commands Status Source Data Help
(): Ins.c:56
Lookup Find» Clear Watch Print Display Plot Hide Rotate Set Undisp
1: Y
2 3 5 9 12 0 0 0 0 0
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
// shift Y[J], Y[J+1],... rightward
// before inserting NewY
ScootOver(J);
Y[J] = NewY;
return;
}
}
void ProcessData()
(gdb) next
(gdb) next
(gdb) next
(gdb) next
(gdb)
Display 1: Y (enabled, scope ProcessData)

Y never changed! The 25 was ignored.

After some thought, we realize that the **for** loop only handles the case in which NewY is inserted within Y. We need separate code for the case in which NewY is added at the right end of Y.

After the **for** loop, add the code:

```
// one more case:  NewY > all existing Y elements
Y[NumY] = NewY;
```

Examining arrays passed as function arguments:

Many people avoid using global variables. Suppose for example the arrays X and Y in our insert-sort example had been declared local to main(). The call to GetArgs(), for instance, would now be

```
GetArgs(X,Argc,Argv);
```

and GetArgs() itself would look like this:

```
void GetArgs(int XX[], int AC, char **AV)

{  int I;

   NumInputs = AC - 1;
   for (I = 0; I < NumInputs; I++)
      XX[I] = atoi(AV[I+1]);
}
```

Suppose we wish to examine XX while in GetArgs(). Our old method of moving the mouse cursor to XX is no longer useful; only the memory address in the pointer XX will be shown.

However, we can still easily view individual array elements, as mentioned earlier. In this context we would:

- Use the mouse to highlight "XX[I]".
- Right-click on the highlighted item.
- Select "Print XX[I]". (Or choose "Display XX[I]" for continuous viewing in a Data window.)
- The value of XX[I] will be printed to the Debugger Console window.

Examining linked data structures:

Some debuggers, including DDD, GVD and JSwat, have nice facilities for displaying linked data structures, which are of great value. To illustrate how this is done in DDD, we've rewritten Ins.c as Lnk.c, which implements Y as a linear linked list instead of as an array:

```
struct ListElt  {
   int Elt;
   struct ListElt *NextElt;
};

struct ListElt *Y;  // ptr to workspace list
```

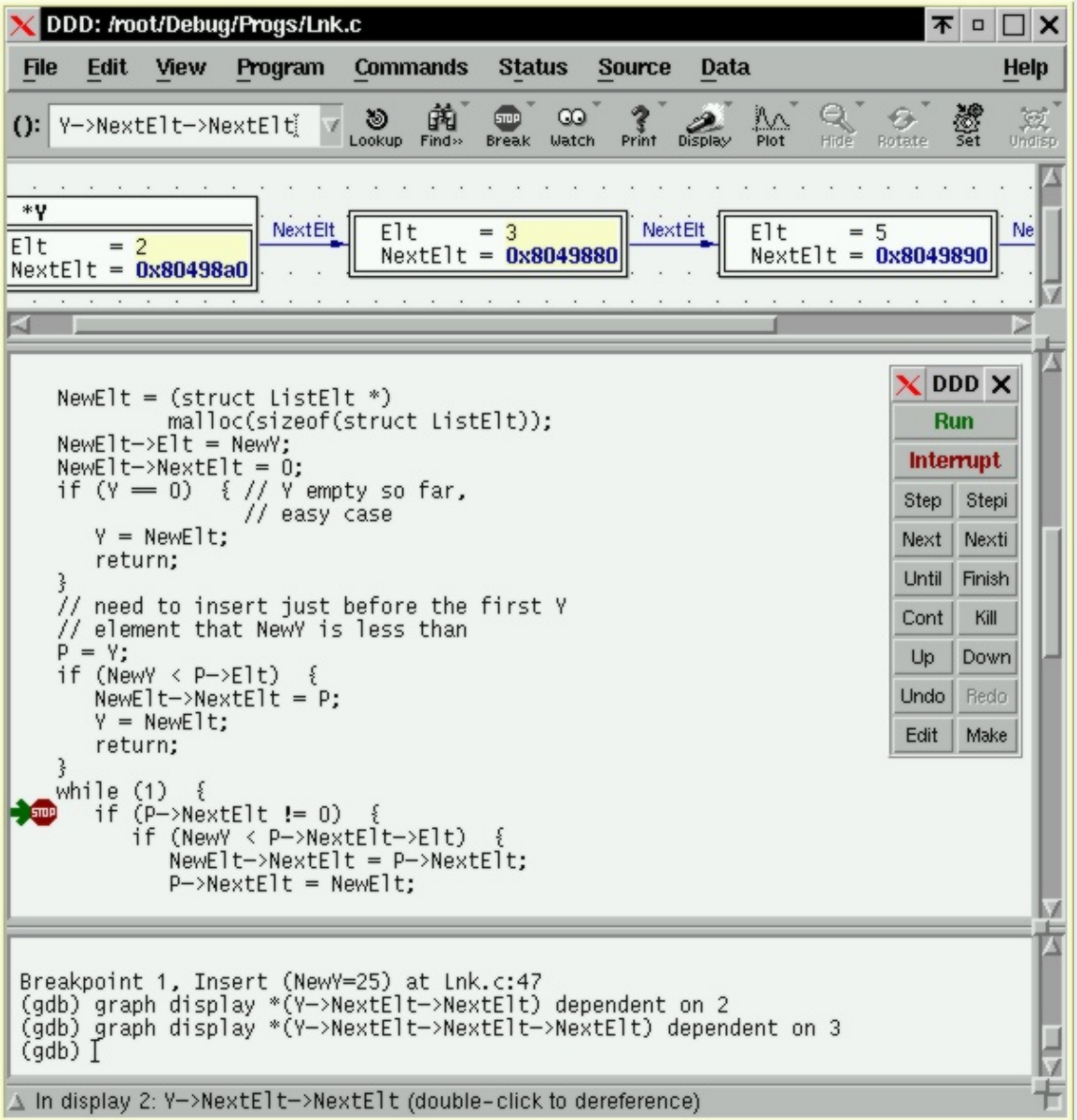
Here is how to display the list:

(a) Right-click anywhere Y appears in the Source File window, and choose “Display *Y”.

(b) A Data window will be created, and a box will appear there showing the contents of the **struct** pointed to by Y.

(c) If NextElt in that **struct** is nonnull, right-click on it, and select “Display *()’.

(d) Repeat step (c) until the entire list (or the portion of interest to you) is displayed.


DDD: /root/Debug/Progs/Lnk.c
File Edit View Program Commands Status Source Data Help
(): Y->NextElt->NextElt
Lookup Find» Break Watch Print Display Plot Hide Rotate Set Undisp
*Y
Elt = 2
NextElt = 0x80498a0
NextElt
Elt = 3
NextElt = 0x8049880
NextElt
Elt = 5
NextElt = 0x8049890
Ne
NewElt = (struct ListElt *)
malloc(sizeof(struct ListElt));
NewElt->Elt = NewY;
NewElt->NextElt = 0;
if (Y == 0) { // Y empty so far,
// easy case
Y = NewElt;
return;
}
// need to insert just before the first Y
// element that NewY is less than
P = Y;
if (NewY < P->Elt) {
NewElt->NextElt = P;
Y = NewElt;
return;
}
while (1) {
if (P->NextElt != 0) {
if (NewY < P->NextElt->Elt) {
NewElt->NextElt = P->NextElt;
P->NextElt = NewElt;
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Breakpoint 1, Insert (NewY=25) at Lnk.c:47
(gdb) graph display *(Y->NextElt->NextElt) dependent on 2
(gdb) graph display *(Y->NextElt->NextElt->NextElt) dependent on 3
(gdb)
In display 2: Y->NextElt->NextElt (double-click to dereference)

Using the display:

- The display will automatically update as you step through the program (except when items are appended, in which case you must add the new link manually).
- If the list is too large for the window, use the scrollbars to view different sections of it.
- To delete part or all of the display, right-click on the displayed item and choose Undisplay.

Summary:

A debugging tool cannot determine what your bug is, but it is great value in determining where it is. You should use one in all of your programming work.

(See related links, next slide.)

Good debugging tools:

My criteria:

- Free (especially helpful for classroom use).
- Advanced features (e.g. display of linked data structures).
- Small memory footprint.
- I prefer standalone debuggers, not integrated development environments (IDEs) — I want to use my own favorite text editor.

Standalone debuggers:

- DDD; featured here; http://heather.cs.ucdavis.edu/~matloff/ddd.html.
- GVD: Very similar to DDD, less general but trimmer, faster-loading; available for both UNIX and Windows; http://heather.cs.ucdavis.edu/~matloff/gvd.html.
- JSwat: Excellent debugger for Java; works on most platforms; can be used as standalone or with the JIPE IDE (see below); http://heather.cs.ucdavis.edu/~matloff/jswat.html.

IDEs:

- KDevelop: For C/C++, on Linux under KDE.
- BlueJ: For Java, especially for learners; works on most platforms; http://heather.cs.ucdavis.edu/~matloff/bluej.html.
- JIPE: For Java; works on most platforms; http://jipe.sourceforge.net/.

Software development is faster and more pleasant if you make good use of a text editor:

- My tutorial on general editing tips for programmers: http://heather.cs.ucdavis.edu/~matloff/progedit.html.
- Vim: Very advanced extension of vi; very popular; multi-platform; free; http://heather.cs.ucdavis.edu/~matloff/vim.html.
- Emacs: A classic; programmable; multi-platform; free; http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/Emacs.html.
- JEdit: Beautiful product; great features; for C/C++/Java/etc; free; http://heather.cs.ucdavis.edu/~matloff/jedit.html.