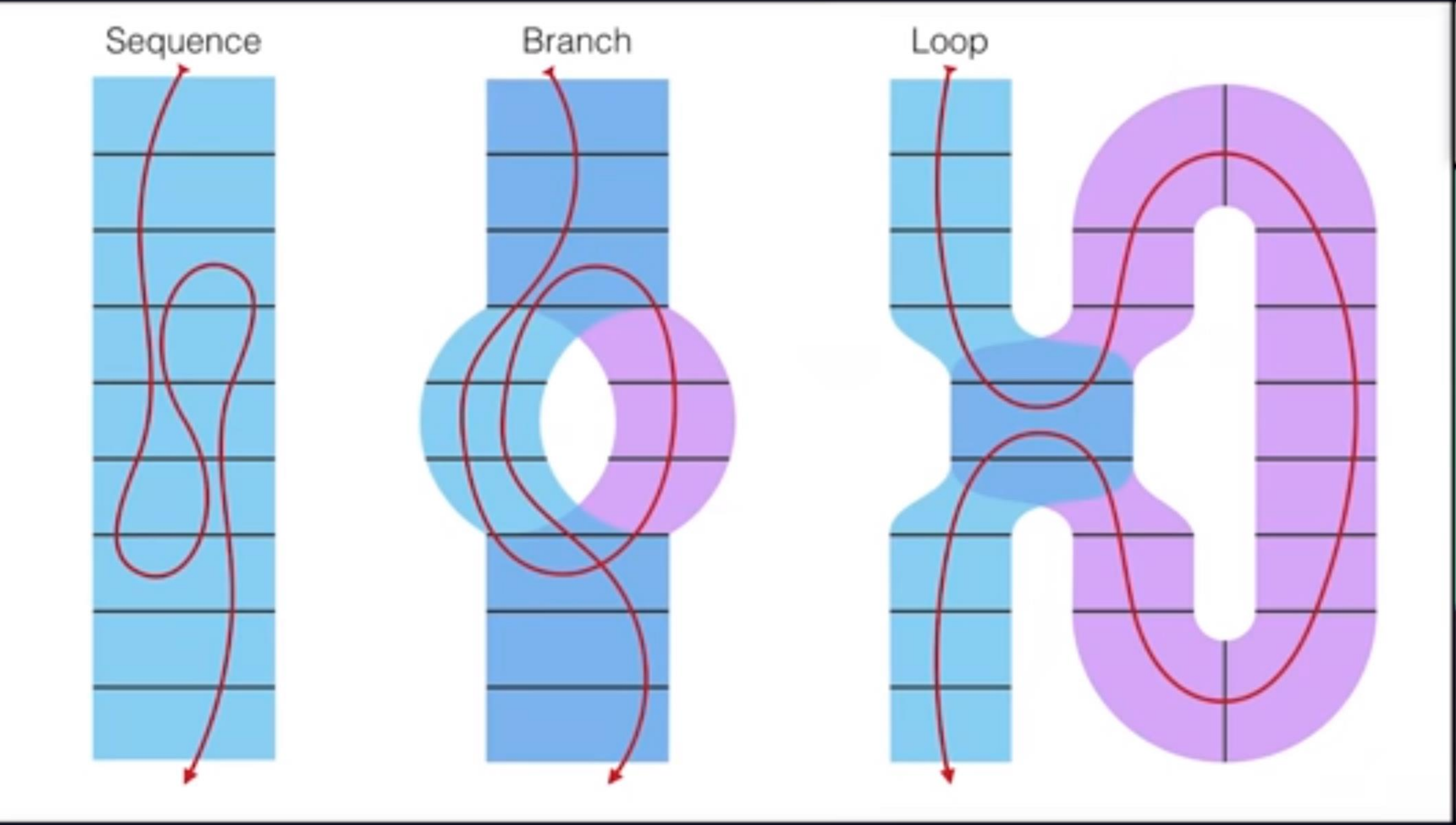
THE SHAPE OF A PROGRAM

KEYNOTE: The Shape of a Program - Lisa Lippincott [ACCU 2018]

conference.accu.org



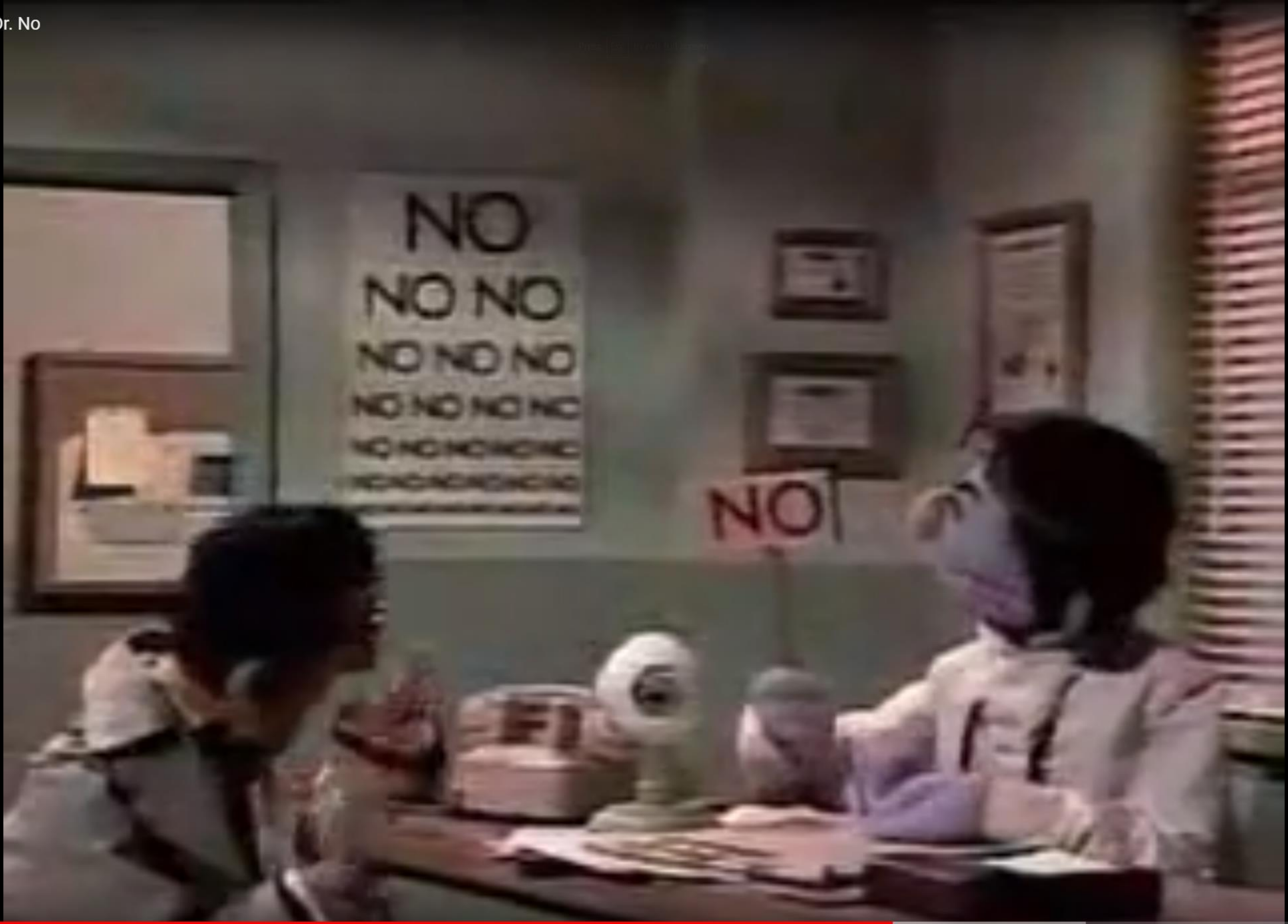
@ACCUconf

11:52 / 59:19

MASTERPIECE
THEATRE

Monsterpiece Theater - The Old Man & The C

Monsterpiece Theater - Dr. No



2:32 / 3:55

Monsterpiece Theater: 1 Flew Over The Cuckoo's Nest
TV Y
2:30 / 3:01

THE SHAPE OF A PROGRAM

```cpp
int main()
{
    // Seed with a real random value, if available
    std::random_device r;

    // Choose a random mean between 1 and 6
    std::default_random_engine e1(r());
    std::uniform_int_distribution<int> uniform_dist(1, 6);
    int mean = uniform_dist(e1);
    std::cout << "Randomly-chosen mean: " << mean << '\n';

    // Generate a normal distribution around that mean
    std::seed_seq seed2{r(), r(), r(), r(), r(), r(), r(), r()};
    std::mt19937 e2(seed2);
    std::normal_distribution<> normal_dist(mean, 2);

    std::map<int, int> hist;
    for (int n = 0; n < 10000; ++n) {
        ++hist[std::round(normal_dist(e2))];
    }
    std::cout << "Normal distribution around " << mean << ":\n";
    for (auto p : hist) {
        std::cout << std::fixed << std::setprecision(1)
            << std::setw(2) << p.first << ' ' <<
            std::string(p.second/200, '*') << '\n';
    }
}
```

(From CppReference)

```
int main()
{
    // Seed with a real random value, if available
    std::random_device r;

    // Choose a random mean between 1 and 6
    std::default_random_engine e1(r());
    std::uniform_int_distribution<int> uniform_dist(1, 6);
    int mean = uniform_dist(e1);
    std::cout << "Randomly-chosen mean: " << mean << '\n';

    // Generate a normal distribution around that mean
    std::seed_seq seed2{r(), r(), r(), r(), r(), r(), r(), r()};
    std::mt19937 e2(seed2);
    std::normal_distribution<> normal_dist(mean, 2);

    std::map<int, int> hist;
    for (int n = 0; n < 10000; ++n) {
        ++hist[std::round(normal_dist(e2))];
    }
    std::cout << "Normal distribution around " << mean << ":\n";
    for (auto p : hist) {
        std::cout << std::fixed << std::setprecision(1)
            << std::setw(2) << p.first << ' ' <<
            std::string(p.second/200, '*') << '\n';
    }
}
```

(From CppReference)

```
void RaiseCoffeeShortage(){ std::array<ULONG_PTR,
2> const Parameters = { (ULONG_PTR)"Oh no! We ran
 out of coffee!", (ULONG_PTR)"Somebody should buy s
ome more!" } }; RaiseException( STATUS_COFFEE_SHOR
TAGE, 0, Parameters.size(), Parameters.data());}in
t MainFilter(EXCEPTION_POINTERS const* Exception,
std::array<char const*, 2>& Parameters){ if (Excep
tion->ExceptionRecord->ExceptionCode != STATUS_COF
FEE_SHORTAGE) { return EXCEPTION_CONTINUE_SEARCH;
} Parameters[0] = (char const*) Exception->Excepti
onRecord->ExceptionInformation[0]; Parameters[1] =
 (char const*) Exception->ExceptionRecord->Excepti
onInformation[1]; return EXCEPTION_EXECUTE_HANDLER
;}void Main(){ std::array<char const*, 2> Exceptio
nParameters{};    try { RaiseCoffeeShortage(); }
except (MainFilter(GetExceptionInformation(), Exce
ptionParameters)) { puts("Uh no! A coffee shortage
 has occurred!"); puts(ExceptionParameters[0]); pu
ts(ExceptionParameters[1]); }}void RaiseCoffeeShor
tage(){ std::array<ULONG_PTR, 2> const Parameters
= { (ULONG_PTR)"Oh no! We ran out of coffee!", (UL
ONG_PTR)"Somebody should buy some more!" } }; Rais
eException( STATUS_COFFEE_SHORTAGE, 0, Parameters.
size(), Parameters.data());}int MainFilter(EXCEPTI
ON_POINTERS const* Exception, std::array<char cons
t*, 2>& Parameters){ if (Exception->ExceptionRecor
d->ExceptionCode != STATUS_COFFEE_SHORTAGE) { retu
rn EXCEPTION_CONTINUE_SEARCH; } Parameters[0] = (c
har const*) Exception->ExceptionRecord->ExceptionI
nformation[0]; Parameters[1] = (char const*) Excep
tion->ExceptionRecord->ExceptionInformation[1]; re
turn EXCEPTION_EXECUTE_HANDLER;}void Main(){ std::
array<char const*, 2> ExceptionParameters{}; try
 { RaiseCoffeeShortage(); } except (MainFilter(G
etExceptionInformation(), ExceptionParameters)) {
puts("Uh no! A coffee shortage has occurred!"); pu
ts(ExceptionParameters[0]); puts(ExceptionParamete
rs[1]); }}void RaiseCoffeeShortage(){ std::array<U
LONG_PTR, 2> const Parameters = { (ULONG_PTR)"Oh
no! We ran out of coffee!", (ULONG_PTR)"Somebody sh
ould buy some more!" } }; RaiseException( STATUS_C
OFFEE_SHORTAGE, 0, Parameters.size(), Parameters.d
ata());}int MainFilter(EXCEPTION_POINTERS const* E
xception, std::array<char const*, 2>& Parameters){
 if (Exception->ExceptionRecord->ExceptionCode !=
STATUS_COFFEE_SHORTAGE) { return EXCEPTION_CONTINU
E_SEARCH; } Parameters[0] = (char const*) Exceptio
n->ExceptionRecord->ExceptionInformation[0]; Param
eters[1] = (char const*) Exception->ExceptionRecor
d->ExceptionInformation[1]; return EXCEPTION_EXECU
```

```c
#include <ncurses.h>/*****************************************************/
            int               m[256                   ] [         256   ],a
 ,b   ;;;   ;;;   WINDOW*w;   char*l=""   "\176qxl"   "q"   "q"   "k"   "w\
xm"   "x"   "t"         "j"         "v"         "u"         "n"         ,Q[
 ]=   "Z"   "pt!ftd`"   "qdc!`eu"   "dq!$c!nnwf"/**   ***   */"t\040\t";c(
int   u ,         int         v){                     v?m   [u]         [v-
 1]   |=2,m[u][v-1] &   48?W][v-1   ] &   15]]):0:0;u?m[u   -1][v]|=1   ,m[
 u-               1][   v]&         48?               W-1   ][v         ]&
15]   ]):0:0;v<   255   ?m[   u][v+1]|=8,m[u][v+1]&   48?   W][   v+1]&15]]
):0         :0;         u <               255   ?m[   u+1         ][v   ]|=
4,m[u+1][   v]&48?W+1][v]&15]]):0:0;W][   v]&   15]   ]);}cu(char*q){   return
 *q               ?cu   (q+         1)&         1?q   [0]               ++:
q[0   ]--   :1;   }d(   int   u ,   int/**/v,   int/**/x,   int   y){   int
Y=y   -v,   X=x         -u;   int         S,s   ;Y<         0?Y   =-Y   ,s,
s=-   1:(   s=1);X<0?X=-X,S   =-1  :(S=   1);   Y<<=   1;X<<=1;   if(X>Y){
int   f=Y               -(X   >>1   );;               while(u!=         x){
f>=   0?v+=s,f-=X:0;u   +=S   ;f+=   Y;m[u][v]|=32;mvwaddch(w,v   ,u,   m[u
 ][               v]&   64?   60:         46)         ;if         (m[   u][
v]&16){c(u,v);;   ;;;   ;;;   return;}}   }else{int   f=X   -(Y>>1);;   while
 (v   !=y         ){f   >=0         ?u    +=S,              f-=         Y:0
 ;v   +=s   ;f+=X;m[u][v]|=   32;mvwaddch(w,v   ,u,m[u][v]&64?60:46);if(m[u
 ][                     v]&         16)   {c(   u,v                     );
  ;   return;;;}}}}Z(   int/**/a,   int   b){   }e(   int/**/y,int/**/  x){
int               i ;         for         (i=         a;i               <=a
+S;i++)d(y,x,i,b),d(y,x,i,b+L);for(i=b;i<=b+L;i++)d(y,x,a,i),d(y,x,a+   S,i
 );                     ;;;         ;;;         ;;;               ;;;   ;
  mvwaddch(w,x,y,64);   ;;;   ;;;   ;;;   prefresh(   w,b,a,0,0   ,L-   1,S-1
);}             main(         int               V ,   char              *C[
  ]   ){FILE*f=   fopen(V==1?"arachnid.c"/**/   :C[   1],"r");int/**/x,y,c,
v=0         ;;;      initscr              ();               Z(Z         (raw
 ()   ,Z(   curs_set(0),Z(1   ,noecho()))),keypad(    stdscr,TRUE));w   =newpad
  (   300,  300               ) ;   for         (x=   255   ; x   >=0   ;x--
  )   for   (y=   255   ;y>=0;y--   )m[   x][   y]=   0;x=y=0;refresh( );while
  (   (c=                           fgetc (f)   )+1)                    {if(
0||c==10||  x==   256){x=0;y++;if(y==256  )break;;}   else{m[x][y]=(c   ==
'~'   ?64   : c   ==32              ?0:         16)   ;;x               ++;
      }}for(x=0   ;x<   256;x++)m   [x][0]=16   ,m[   x][   255]=16;for(y=0
;y<         256   ; y         ++)   m[0         ][y   ] =               16,
m[255][y]   =16   ;a=b=c=0;   x=y   =1;   do{v++;mvwaddch   (w,   y,x   ,m[
x][         y]&               32?   m[x                     ][y   ] &   16?
 0|   acs_map[l[m[x][y]&15]]:46 :   32);c==0163&&!(m[x][y+1]&16)?y++:   0;c
 ==   119         &&!         (m[                                       x][
 y-   1]&   16)   ?y--:0;;c   ==97  &&!(m[x-1][y]&16)?x--:0;c==100&&!(m[x+1
 ][   y]&   16)         ? x   ++:0              ;if(                    c==
 3-   1+1   ){endwin(   );;   return(0)   ;}x   -a<5?a>S-   5?a-=S-5:(a=0):
0;x               -a>         S-5?a<255   -S*         2?a               +=S
-5:(a=256-S):0;   y-b<5?b>L-5?b-=L-5:(b   =0)   :0;   y-b>L-5?b<255-L   *2?
b+=                                       L-5   :(b                     =256
-L)   :0;e(x,y);if(m[x][y]&64)break;}while((c=getch())!=-1);endwin();cu(Q);
printf(Q,v);}
```

From Nick Johnson,
2004 IOCCC Winner

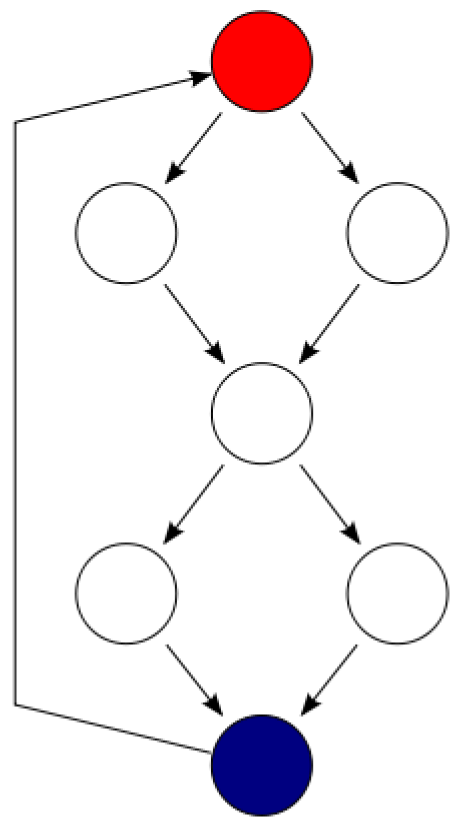
MAYBE I SHOULD SHARE SOMETHING USEFUL

LET’S TALK ABOUT PROGRAM COMPLEXITY

```c
int main()
{
    if (c1())
    {
        f1();
    }
    else
    {
        f2();
    }

    if (c2())
    {
        f3();
    }
    else
    {
        f4();
    }
}
```

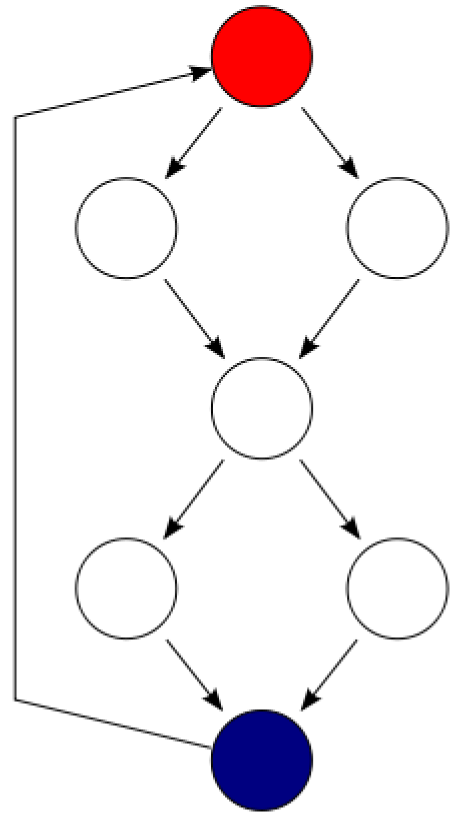
(From Wikipedia)

```c
int main()
{
    if (c1())
    {
        f1();
    }
    else
    {
        f2();
    }

    if (c2())
    {
        f3();
    }
    else
    {
        f4();
    }
}
```



(From Wikipedia)

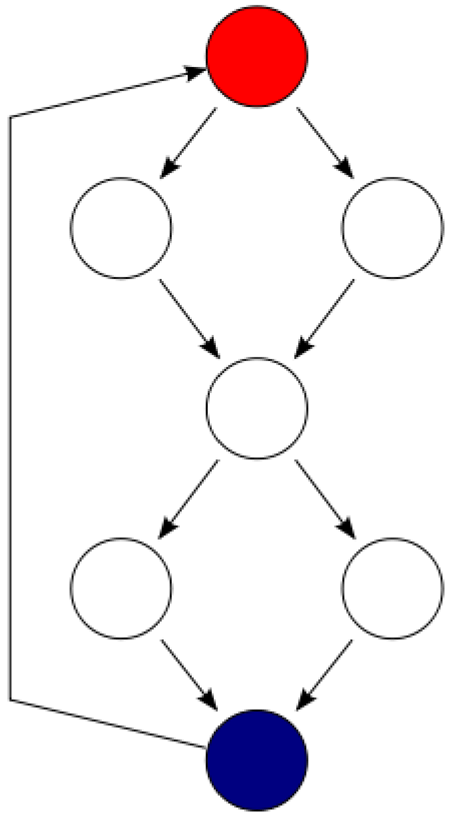
```
int main()
{
    if (c1())
    {
        f1();
    }
    else
    {
        f2();
    }

    if (c2())
    {
        f3();
    }
    else
    {
        f4();
    }
}
```



E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

(From Wikipedia)

```
int main()
{
    if (c1())
    {
        f1();
    }
    else
    {
        f2();
    }

    if (c2())
    {
        f3();
    }
    else
    {
        f4();
    }
}
```

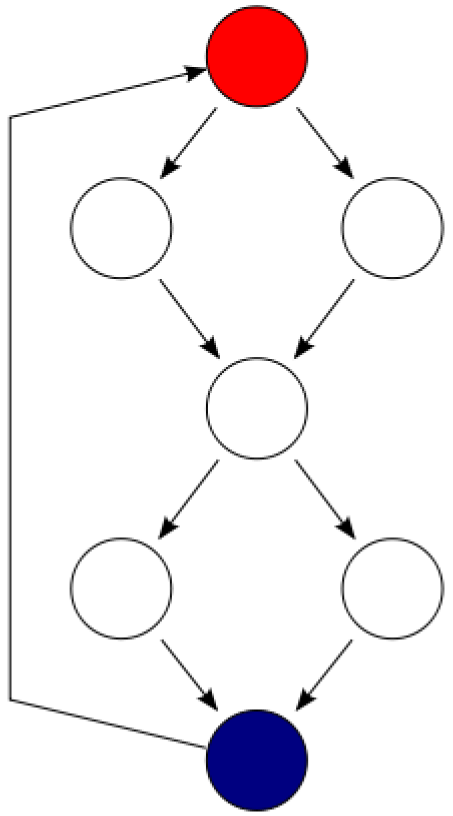


E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

$$M = E - N + P$$

(From Wikipedia)

```
int main()
{
    if (c1())
    {
        f1();
    }
    else
    {
        f2();
    }

    if (c2())
    {
        f3();
    }
    else
    {
        f4();
    }
}
```



E = the number of edges of the graph.
N = the number of nodes of the graph.
P = the number of connected components.

$$M = E - N + P$$

This Program Is

3

(From Wikipedia)

THREE SHALL BE THE NUMBER OF THE PROGRAM...
...AND THE NUMBER OF THE PROGRAM SHALL BE THREE

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

(From Wikipedia)

- Program vocabulary: $\eta = \eta_1 + \eta_2$
- Program length: $N = N_1 + N_2$
- Calculated program length: $\hat{N} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume: $V = N \times \log_2 \eta$
- Difficulty : $D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort: $E = D \times V$

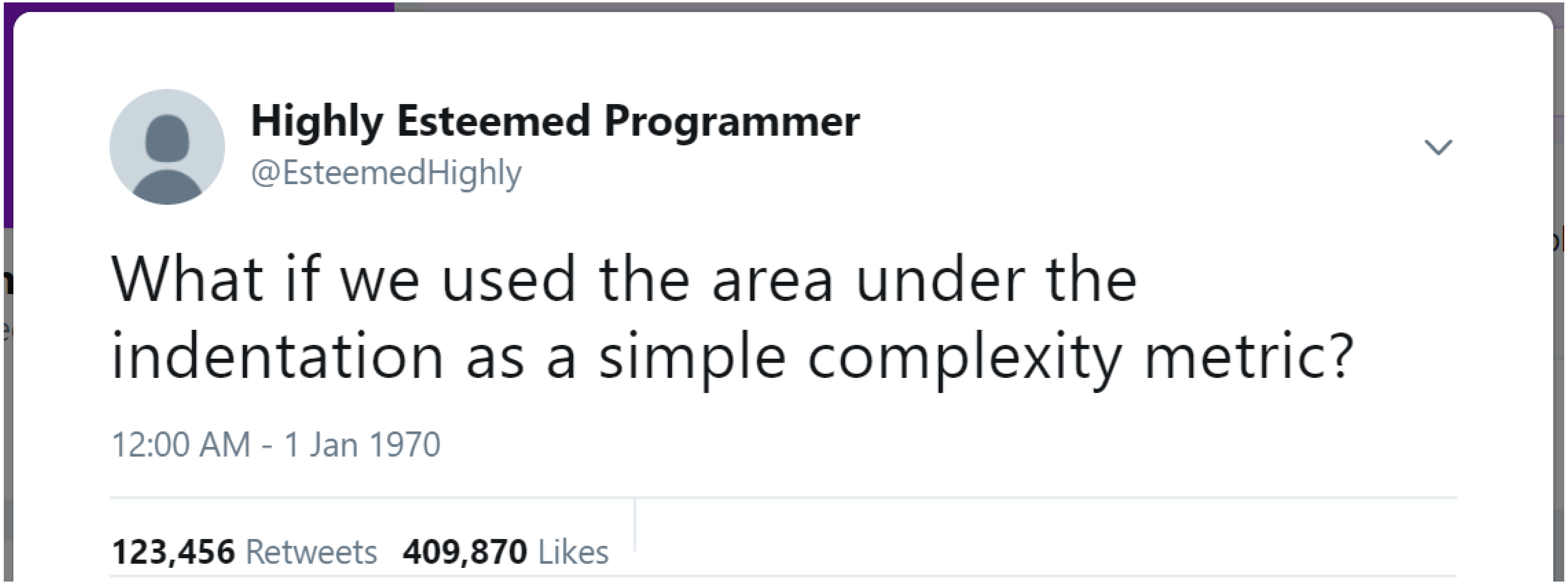
- $\eta_1 = 12$, $\eta_2 = 7$, $\eta = 19$
- $N_1 = 27$, $N_2 = 15$, $N = 42$
- Calculated Estimated Program Length: $\hat{N} = 12 \times log_2 12 + 7 \times log_2 7 = 62.67$
- Volume: $V = 42 \times log_2 19 = 178.4$
- Difficulty: $D = \frac{12}{2} \times \frac{15}{7} = 12.85$
- Effort: $E = 12.85 \times 178.4 = 2292.44$
- Time required to program: $T = \frac{2292.44}{18} = 127.357$ seconds
- Number of delivered bugs: $B = \frac{2292.44^{\frac{2}{3}}}{3000} = 0.05$

(From Wikipedia)

Many measures of software complexity have been proposed.

Many of these, although yielding a good representation of complexity, do not lend themselves to easy measurement.

(From Wikipedia)



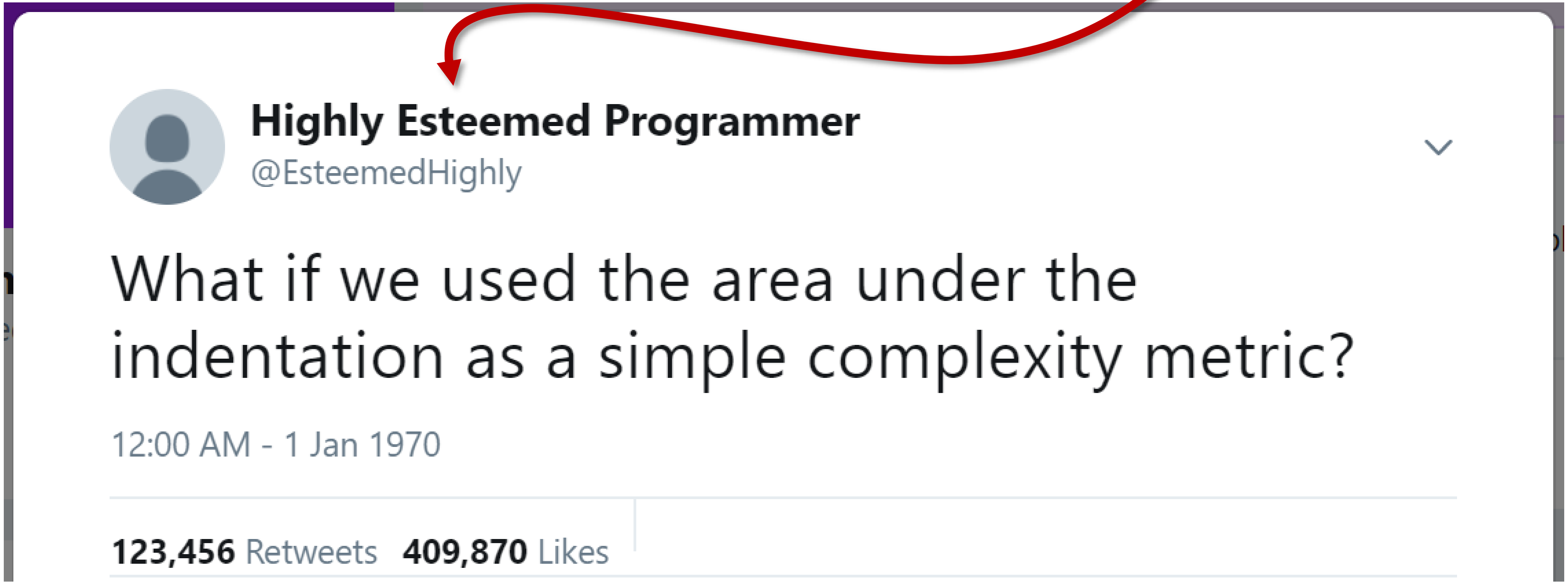
Highly Esteemed Programmer
@EsteemedHighly

What if we used the area under the indentation as a simple complexity metric?

12:00 AM - 1 Jan 1970

123,456 Retweets **409,870** Likes

I… forgot who tweeted this



Highly Esteemed Programmer
@EsteemedHighly

What if we used the area under the indentation as a simple complexity metric?

12:00 AM - 1 Jan 1970

123,456 Retweets **409,870** Likes

```cpp
int main()
{
    // Seed with a real random value, if available
    std::random_device r;

    // Choose a random mean between 1 and 6
    std::default_random_engine e1(r());
    std::uniform_int_distribution<int> uniform_dist(1, 6);
    int mean = uniform_dist(e1);
    std::cout << "Randomly-chosen mean: " << mean << '\n';

    // Generate a normal distribution around that mean
    std::seed_seq seed2{r(), r(), r(), r(), r(), r(), r(), r()};
    std::mt19937 e2(seed2);
    std::normal_distribution<> normal_dist(mean, 2);

    std::map<int, int> hist;
    for (int n = 0; n < 10000; ++n) {
        ++hist[std::round(normal_dist(e2))];
    }
    std::cout << "Normal distribution around " << mean << ":\n";
    for (auto p : hist) {
        std::cout << std::fixed << std::setprecision(1)
            << std::setw(2) << p.first << ' ' <<
            std::string(p.second/200, '*') << '\n';
    }
}
```

(From CppReference)

```cpp
int main()
{
    // Seed with a real random value, if available
    std::random_device r;

    // Choose a random mean between 1 and 6
    std::default_random_engine e1(r());
    std::uniform_int_distribution<int> uniform_dist(1, 6);
    int mean = uniform_dist(e1);
    std::cout << "Randomly-chosen mean: " << mean << '\n';

    // Generate a normal distribution around that mean
    std::seed_seq seed2{r(), r(), r(), r(), r(), r(), r(), r()};
    std::mt19937 e2(seed2);
    std::normal_distribution<> normal_dist(mean, 2);

    std::map<int, int> hist;
    for (int n = 0; n < 10000; ++n) {
        ++hist[std::round(normal_dist(e2))];
    }
    std::cout << "Normal distribution around " << mean << ":\n";
    for (auto p : hist) {
        std::cout << std::fixed << std::setprecision(1)
            << std::setw(2) << p.first << ' ' <<
            std::string(p.second/200, '*') << '\n';
    }
}
```

(From CppReference)

```cpp
void print_nonzero_elements(std::vector<std::vector<int>> v)
{
    for (auto outerIt = v.begin(); outerIt != v.end(); ++outerIt)
    {
        for (auto innerIt = outerIt->begin(); innerIt != outerIt->end(); ++innerIt)
        {
            if (*innerIt != 0)
            {
                std::cout << *innerIt << ' ';
            }
        }

        std::cout << '\n';
    }
}
```

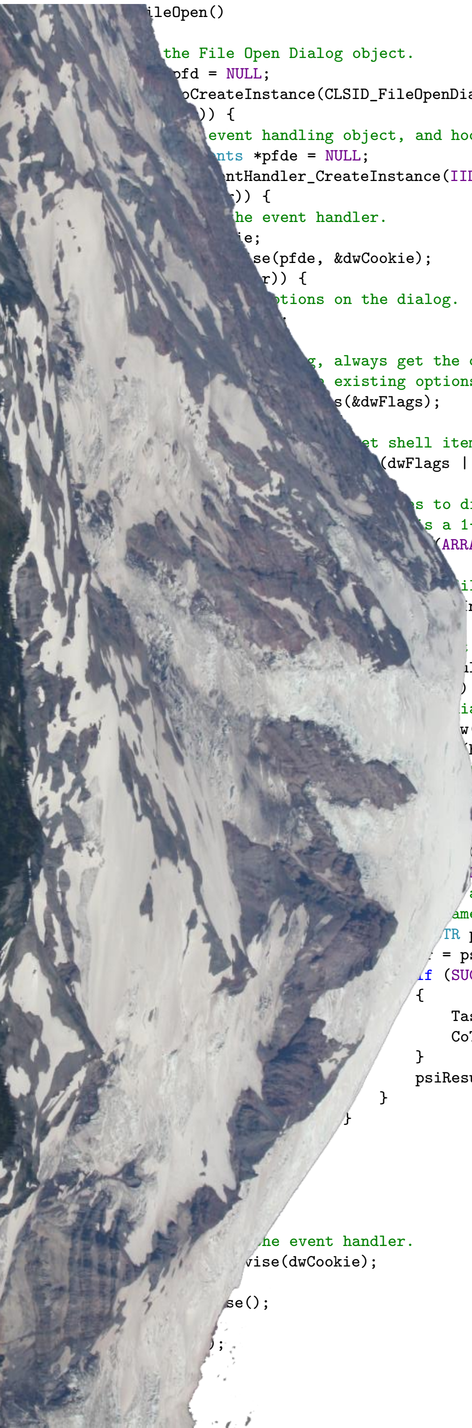
```
void print_nonzero_elements(std::vector<std::vector<int>> v)
{
    for (auto outerIt = v.begin(); outerIt != v.end(); ++outerIt)
    {
        for (auto innerIt = outerIt->begin(); innerIt != outerIt->end(); ++innerIt)
        {
            if (*innerIt != 0)
            {
                std::cout << *innerIt << ' ';
            }
        }

        std::cout << '\n';
    }
}
```

```
HRESULT BasicFileOpen()
{
    // CoCreate the File Open Dialog object.
    IFileDialog *pfd = NULL;
    HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
    if (SUCCEEDED(hr)) {
        // Create an event handling object, and hook it up to the dialog.
        IFileDialogEvents *pfde = NULL;
        hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
        if (SUCCEEDED(hr)) {
            // Hook up the event handler.
            DWORD dwCookie;
            hr = pfd->Advise(pfde, &dwCookie);
            if (SUCCEEDED(hr)) {
                // Set the options on the dialog.
                DWORD dwFlags;

                // Before setting, always get the options first in order
                // not to override existing options.
                hr = pfd->GetOptions(&dwFlags);
                if (SUCCEEDED(hr)) {
                    // In this case, get shell items only for file system items.
                    hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
                    if (SUCCEEDED(hr)) {
                        // Set the file types to display only.
                        // Notice that this is a 1-based array.
                        hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
                        if (SUCCEEDED(hr)) {
                            // Set the selected file type index to Word Docs for this example.
                            hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
                            if (SUCCEEDED(hr)) {
                                // Set the default extension to be ".doc" file.
                                hr = pfd->SetDefaultExtension(L"doc;docx");
                                if (SUCCEEDED(hr)) {
                                    // Show the dialog
                                    hr = pfd->Show(NULL);
                                    if (SUCCEEDED(hr)) {
                                        // Obtain the result once the user clicks
                                        // the 'Open' button.
                                        // The result is an IShellItem object.
                                        IShellItem *psiResult;
                                        hr = pfd->GetResult(&psiResult);
                                        if (SUCCEEDED(hr)) {
                                            // We are just going to print out the
                                            // name of the file for sample sake.
                                            PWSTR pszFilePath = NULL;
                                            hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                                            if (SUCCEEDED(hr))
                                            {
                                                TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                                                CoTaskMemFree(pszFilePath);
                                            }
                                            psiResult->Release();
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                // Unhook the event handler.
                pfd->Unadvise(dwCookie);
            }
            pfde->Release();
        }
        pfd->Release();
    }
    return hr;
}
```

(From MSDN)

```
HRESULT BasicFileOpen()
{
    // CoCreate the File Open Dialog object.
    IFileDialog *pfd = NULL;
    HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
    if (SUCCEEDED(hr)) {
        // Create an event handling object, and hook it up to the dialog.
        IFileDialogEvents *pfde = NULL;
        hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
        if (SUCCEEDED(hr)) {
            // Hook up the event handler.
            DWORD dwCookie;
            hr = pfd->Advise(pfde, &dwCookie);
            if (SUCCEEDED(hr)) {
                // Set the options on the dialog.
                DWORD dwFlags;

                // Before setting, always get the options first in order
                // not to override existing options.
                hr = pfd->GetOptions(&dwFlags);
                if (SUCCEEDED(hr)) {
                    // In this case, get shell items only for file system items.
                    hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
                    if (SUCCEEDED(hr)) {
                        // Set the file types to display only.
                        // Notice that this is a 1-based array.
                        hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
                        if (SUCCEEDED(hr)) {
                            // Set the selected file type index to Word Docs for this example.
                            hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
                            if (SUCCEEDED(hr)) {
                                // Set the default extension to be ".doc" file.
                                hr = pfd->SetDefaultExtension(L"doc;docx");
                                if (SUCCEEDED(hr)) {
                                    // Show the dialog
                                    hr = pfd->Show(NULL);
                                    if (SUCCEEDED(hr)) {
                                        // Obtain the result once the user clicks
                                        // the 'Open' button.
                                        // The result is an IShellItem object.
                                        IShellItem *psiResult;
                                        hr = pfd->GetResult(&psiResult);
                                        if (SUCCEEDED(hr)) {
                                            // We are just going to print out the
                                            // name of the file for sample sake.
                                            PWSTR pszFilePath = NULL;
                                            hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                                            if (SUCCEEDED(hr))
                                            {
                                                TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                                                CoTaskMemFree(pszFilePath);
                                            }
                                            psiResult->Release();
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                // Unhook the event handler.
                pfd->Unadvise(dwCookie);
            }
            pfde->Release();
        }
        pfd->Release();
    }
    return hr;
}
```

(From MSDN)



```
                ileOpen()

                  the File Open Dialog object.
                   pfd = NULL;
                    oCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
                      )) {
                       event handling object, and hook it up to the dialog.
                        nts *pfde = NULL;
                         ntHandler_CreateInstance(IID_PPV_ARGS(&pfde));
                          )) {
                          he event handler.
                           ie;
                            se(pfde, &dwCookie);
                             r)) {
                              tions on the dialog.

                                  g, always get the options first in order
                                     existing options.
                                     s(&dwFlags);

                                        et shell items only for file system items.
                                          (dwFlags | FOS_FORCEFILESYSTEM);

                                             s to display only.
                                               s a 1-based array.
                                                (ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);

                                                  ile type index to Word Docs for this example.
                                                   ndex(INDEX_WORDDOC);

                                                    extension to be ".doc" file.
                                                   ltExtension(L"doc;docx");
                                                   ) {
                                                   ialog
                                                   w(NULL);
                                                   (hr)) {
                                                    the result once the user clicks
                                                    n' button.
                                                    lt is an IShellItem object.
                                                     *psiResult;
                                                    GetResult(&psiResult);
                                                    DED(hr)) {
                                                    are just going to print out the
                                                  ame of the file for sample sake.
                                                 TR pszFilePath = NULL;
                                                 = psiResult->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                                              if (SUCCEEDED(hr))
                                              {
                                                  TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                                                  CoTaskMemFree(pszFilePath);
                                              }
                                              psiResult->Release();
                                          }
                                      }




                            he event handler.
                           vise(dwCookie);

                         se();

                       );
```

```
HRESULT BasicFileOpen()
{
    // CoCreate the File Open Dialog object.
    IFileDialog *pfd = NULL;
    HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
    if (SUCCEEDED(hr)) {
        // Create an event handling object, and hook it up to the dialog.
        IFileDialogEvents *pfde = NULL;
        hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
        if (SUCCEEDED(hr)) {
            // Hook up the event handler.
            DWORD dwCookie;
            hr = pfd->Advise(pfde, &dwCookie);
            if (SUCCEEDED(hr)) {
                // Set the options on the dialog.
                DWORD dwFlags;

                // Before setting, always get the options first in order
                // not to override existing options.
                hr = pfd->GetOptions(&dwFlags);
                if (SUCCEEDED(hr)) {
                    // In this case, get shell items only for file system items.
                    hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
                    if (SUCCEEDED(hr)) {
                        // Set the file types to display only.
                        // Notice that this is a 1-based array.
                        hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
                        if (SUCCEEDED(hr)) {
                            // Set the selected file type index to Word Docs for this example.
                            hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
                            if (SUCCEEDED(hr)) {
                                // Set the default extension to be ".doc" file.
                                hr = pfd->SetDefaultExtension(L"doc;docx");
                                if (SUCCEEDED(hr)) {
                                    // Show the dialog
                                    hr = pfd->Show(NULL);
                                    if (SUCCEEDED(hr)) {
                                        // Obtain the result once the user clicks
                                        // the 'Open' button.
                                        // The result is an IShellItem object.
                                        IShellItem *psiResult;
                                        hr = pfd->GetResult(&psiResult);
                                        if (SUCCEEDED(hr)) {
                                            // We are just going to print out the
                                            // name of the file for sample sake.
                                            PWSTR pszFilePath = NULL;
                                            hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH, &pszFilePath);
                                            if (SUCCEEDED(hr))
                                            {
                                                TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                                                CoTaskMemFree(pszFilePath);
                                            }
                                            psiResult->Release();
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
                // Unhook the event handler.
                pfd->Unadvise(dwCookie);
            }
            pfde->Release();
        }
        pfd->Release();
    }
    return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
    // Hook up the event handler.
    DWORD dwCookie;
    hr = pfd->Advise(pfde, &dwCookie);
    if (SUCCEEDED(hr)) {
        // Set the options on the dialog.
        DWORD dwFlags;

        // Before setting, always get the options first in order
        // not to override existing options.
        hr = pfd->GetOptions(&dwFlags);
        if (SUCCEEDED(hr)) {
            // In this case, get shell items only for file system items.
            hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
            if (SUCCEEDED(hr)) {
                // Set the file types to display only.
                // Notice that this is a 1-based array.
                hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
                if (SUCCEEDED(hr)) {
                    // Set the selected file type index to Word Docs for this example.
                    hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
                    if (SUCCEEDED(hr)) {
                        // Set the default extension to be ".doc" file.
                        hr = pfd->SetDefaultExtension(L"doc;docx");
                        if (SUCCEEDED(hr)) {
                            // Show the dialog
                            hr = pfd->Show(NULL);
                            if (SUCCEEDED(hr)) {
                                // Obtain the result once the user clicks
                                // the 'Open' button.
                                // The result is an IShellItem object.
                                IShellItem *psiResult;
                                hr = pfd->GetResult(&psiResult);
                                if (SUCCEEDED(hr)) {
                                    // We are just going to print out the
                                    // name of the file for sample sake.
                                    PWSTR pszFilePath = NULL;
                                    hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH,
                                                        &pszFilePath);
                                    if (SUCCEEDED(hr))
                                    {
                                        TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                                        CoTaskMemFree(pszFilePath);
                                    }
                                    psiResult->Release();
                                }
                            }
                        }
                    }
                }
            }
        }
        // Unhook the event handler.
        pfd->Unadvise(dwCookie);
    }
    pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
// Hook up the event handler.
DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (SUCCEEDED(hr)) {
// Set the options on the dialog.
DWORD dwFlags;

// Before setting, always get the options first in order
// not to override existing options.
hr = pfd->GetOptions(&dwFlags);
if (SUCCEEDED(hr)) {
    // In this case, get shell items only for file system items.
    hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
    if (SUCCEEDED(hr)) {
        // Set the file types to display only.
        // Notice that this is a 1-based array.
        hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
        if (SUCCEEDED(hr)) {
            // Set the selected file type index to Word Docs for this example.
            hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
            if (SUCCEEDED(hr)) {
                // Set the default extension to be ".doc" file.
                hr = pfd->SetDefaultExtension(L"doc;docx");
                if (SUCCEEDED(hr)) {
                    // Show the dialog
                    hr = pfd->Show(NULL);
                    if (SUCCEEDED(hr)) {
                        // Obtain the result once the user clicks
                        // the 'Open' button.
                        // The result is an IShellItem object.
                        IShellItem *psiResult;
                        hr = pfd->GetResult(&psiResult);
                        if (SUCCEEDED(hr)) {
                            // We are just going to print out the
                            // name of the file for sample sake.
                            PWSTR pszFilePath = NULL;
                            hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH,
                                                 &pszFilePath);
                            if (SUCCEEDED(hr))
                            {
                                TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                                CoTaskMemFree(pszFilePath);
                            }
                            psiResult->Release();
                        }
                    }
                }
            }
        }
    }
}
// Unhook the event handler.
pfd->Unadvise(dwCookie);
}
pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
// Hook up the event handler.
DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (SUCCEEDED(hr)) {
// Set the options on the dialog.
DWORD dwFlags;

// Before setting, always get the options first in order
// not to override existing options.
hr = pfd->GetOptions(&dwFlags);
if (SUCCEEDED(hr)) {
// In this case, get shell items only for file system items.
hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
if (SUCCEEDED(hr)) {
// Set the file types to display only.
// Notice that this is a 1-based array.
hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
if (SUCCEEDED(hr)) {
    // Set the selected file type index to Word Docs for this example.
    hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
    if (SUCCEEDED(hr)) {
        // Set the default extension to be ".doc" file.
        hr = pfd->SetDefaultExtension(L"doc;docx");
        if (SUCCEEDED(hr)) {
            // Show the dialog
            hr = pfd->Show(NULL);
            if (SUCCEEDED(hr)) {
                // Obtain the result once the user clicks
                // the 'Open' button.
                // The result is an IShellItem object.
                IShellItem *psiResult;
                hr = pfd->GetResult(&psiResult);
                if (SUCCEEDED(hr)) {
                    // We are just going to print out the
                    // name of the file for sample sake.
                    PWSTR pszFilePath = NULL;
                    hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH,
                                        &pszFilePath);
                    if (SUCCEEDED(hr))
                    {
                        TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                        CoTaskMemFree(pszFilePath);
                    }
                    psiResult->Release();
                }
            }
        }
    }
}
}
}
// Unhook the event handler.
pfd->Unadvise(dwCookie);
}
pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
// Hook up the event handler.
DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (SUCCEEDED(hr)) {
// Set the options on the dialog.
DWORD dwFlags;

// Before setting, always get the options first in order
// not to override existing options.
hr = pfd->GetOptions(&dwFlags);
if (SUCCEEDED(hr)) {
// In this case, get shell items only for file system items.
hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
if (SUCCEEDED(hr)) {
// Set the file types to display only.
// Notice that this is a 1-based array.
hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
if (SUCCEEDED(hr)) {
// Set the selected file type index to Word Docs for this example.
hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
if (SUCCEEDED(hr)) {
// Set the default extension to be ".doc" file.
hr = pfd->SetDefaultExtension(L"doc;docx");
if (SUCCEEDED(hr)) {
    // Show the dialog
    hr = pfd->Show(NULL);
    if (SUCCEEDED(hr)) {
        // Obtain the result once the user clicks
        // the 'Open' button.
        // The result is an IShellItem object.
        IShellItem *psiResult;
        hr = pfd->GetResult(&psiResult);
        if (SUCCEEDED(hr)) {
            // We are just going to print out the
            // name of the file for sample sake.
            PWSTR pszFilePath = NULL;
            hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH,
                                &pszFilePath);
            if (SUCCEEDED(hr))
            {
                TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
                CoTaskMemFree(pszFilePath);
            }
            psiResult->Release();
        }
    }
}
}
}
}
}
// Unhook the event handler.
pfd->Unadvise(dwCookie);
}
pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
// Hook up the event handler.
DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (SUCCEEDED(hr)) {
// Set the options on the dialog.
DWORD dwFlags;

// Before setting, always get the options first in order
// not to override existing options.
hr = pfd->GetOptions(&dwFlags);
if (SUCCEEDED(hr)) {
// In this case, get shell items only for file system items.
hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
if (SUCCEEDED(hr)) {
// Set the file types to display only.
// Notice that this is a 1-based array.
hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
if (SUCCEEDED(hr)) {
// Set the selected file type index to Word Docs for this example.
hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
if (SUCCEEDED(hr)) {
// Set the default extension to be ".doc" file.
hr = pfd->SetDefaultExtension(L"doc;docx");
if (SUCCEEDED(hr)) {
// Show the dialog
hr = pfd->Show(NULL);
if (SUCCEEDED(hr)) {
// Obtain the result once the user clicks
// the 'Open' button.
// The result is an IShellItem object.
IShellItem *psiResult;
hr = pfd->GetResult(&psiResult);
if (SUCCEEDED(hr)) {
    // We are just going to print out the
    // name of the file for sample sake.
    PWSTR pszFilePath = NULL;
    hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH,
                        &pszFilePath);
    if (SUCCEEDED(hr))
    {
        TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
        CoTaskMemFree(pszFilePath);
    }
    psiResult->Release();
}
}
}
}
}
}
}
// Unhook the event handler.
pfd->Unadvise(dwCookie);
}
pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
// Hook up the event handler.
DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (SUCCEEDED(hr)) {
// Set the options on the dialog.
DWORD dwFlags;

// Before setting, always get the options first in order
// not to override existing options.
hr = pfd->GetOptions(&dwFlags);
if (SUCCEEDED(hr)) {
// In this case, get shell items only for file system items.
hr = pfd->SetOptions(dwFlags | FOS_FORCEFILESYSTEM);
if (SUCCEEDED(hr)) {
// Set the file types to display only.
// Notice that this is a 1-based array.
hr = pfd->SetFileTypes(ARRAYSIZE(c_rgSaveTypes), c_rgSaveTypes);
if (SUCCEEDED(hr)) {
// Set the selected file type index to Word Docs for this example.
hr = pfd->SetFileTypeIndex(INDEX_WORDDOC);
if (SUCCEEDED(hr)) {
// Set the default extension to be ".doc" file.
hr = pfd->SetDefaultExtension(L"doc;docx");
if (SUCCEEDED(hr)) {
// Show the dialog
hr = pfd->Show(NULL);
if (SUCCEEDED(hr)) {
// Obtain the result once the user clicks
// the 'Open' button.
// The result is an IShellItem object.
IShellItem *psiResult;
hr = pfd->GetResult(&psiResult);
if (SUCCEEDED(hr)) {
// We are just going to print out the
// name of the file for sample sake.
PWSTR pszFilePath = NULL;
hr = psiResult->GetDisplayName(SIGDN_FILESYSPATH,
&pszFilePath);
if (SUCCEEDED(hr))
{
TaskDialog(NULL, NULL, L"CommonFileDialogApp", pszFilePath, NULL, TDCBF_OK_BUTTON, TD_INFORMATION_ICON, NULL);
CoTaskMemFree(pszFilePath);
}
psiResult->Release();
}
}
}
}
}
}
}
// Unhook the event handler.
pfd->Unadvise(dwCookie);
}
pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```
HRESULT BasicFileOpen()
{
// CoCreate the File Open Dialog object.
IFileDialog *pfd = NULL;
HRESULT hr = CoCreateInstance(CLSID_FileOpenDialog, NULL, CLSCTX_INPROC_SERVER, IID_PPV_ARGS(&pfd));
if (SUCCEEDED(hr)) {
// Create an event handling object, and hook it up to the dialog.
IFileDialogEvents *pfde = NULL;
hr = CDialogEventHandler_CreateInstance(IID_PPV_ARGS(&pfde));
if (SUCCEEDED(hr)) {
// Hook up the event handler.
DWORD dwCookie;
hr = pfd->Advise(pfde, &dwCookie);
if (SUCCEEDED(hr)) {
// Set the o           the dialog.
DWORD dwFl

// Before               t the options first in order
// not to ov                tions.
hr = pfd->GetOp
if (SUCCEEDED(hr)
// In this case, ge               ly for file system items.
hr = pfd->SetOptions                RCEFILESYSTEM);
if (SUCCEEDED(hr)) {
// Set the file types to
// Notice that this is a
hr = pfd->SetFileTypes(ARRA                 es), c_rgSaveType
if (SUCCEEDED(hr)) {
// Set the selected file type i              for this
hr = pfd->SetFileTypeIndex(INDEX_
if (SUCCEEDED(hr)) {
// Set the default extension to be "
hr = pfd->SetDefaultExtension(L"doc;do
if (SUCCEEDED(hr)) {
// Show the dialog
hr = pfd->Show(NULL);
if (SUCCEEDED(hr)) {
// Obtain the result once the user clicks
// the 'Open' button.
// The result is an IShellItem object
IShellItem *psiResult;
hr = pfd->GetResult(&psiResult);
if (SUCCEEDED(hr)) {
// We are just going to print
// name of the file for sam
PWSTR pszFilePath = NULL;
hr = psiResult->GetDisp                SYSPATH,
&pszFilePath);
if (SUCCEEDED(hr))
{
TaskDialog(NULL            leDialogApp", pszFilePath, NULL, T            TD_INFORMATION_ICON, NULL);
CoTaskMemFree
}
psiResult
}
}
}
}
}
}
}
// Unhook the event handler.
pfd->Unadvise(dwCookie);
}
pfde->Release();
}
pfd->Release();
}
return hr;
}
```

```cpp
void DoThing(int index)
{
    if (IsValidIndexOfOtherThing(index))
    {
        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (IsValidIndexOfOtherThing(index))
    {
        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
    }
}
```

```
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
    }
}
```

```
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
}
```

```
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (CanDoSomethingWithNumber(index))
    {
        if (CheckSomethingCriticalAboutValue(index))
        {
            for (auto const& value : GetData(index))
            {
                switch (value % 3)
                {
                case 0:
                    PrintFoo(value);
                    break;

                case 1:
                    PrintBar(value);
                    break;

                case 2:
                    PrintBaz(value);
                    break;
                }
            }
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        switch (value % 3)
        {
        case 0:
            PrintFoo(value);
            break;

        case 1:
            PrintBar(value);
            break;

        case 2:
            PrintBaz(value);
            break;
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        switch (value % 3)
        {
        case 0:
            PrintFoo(value);
            break;

        case 1:
            PrintBar(value);
            break;

        case 2:
            PrintBaz(value);
            break;
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (IsValidIndexOfOtherThing(index))
    {
        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        switch (value % 3)
        {
        case 0:
            PrintFoo(value);
            break;

        case 1:
            PrintBar(value);
            break;

        case 2:
            PrintBaz(value);
            break;
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        switch (value % 3)
        {
        case 0:
            PrintFoo(value);
            break;

        case 1:
            PrintBar(value);
            break;

        case 2:
            PrintBaz(value);
            break;
        }
    }
}
```

```
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
                                                                              switch (value % 3)
                                                                              {
                                                                              case 0:
                                                                                  PrintFoo(value);
                                                                                  break;

                                                                              case 1:
                                                                                  PrintBar(value);
                                                                                  break;

                                                                              case 2:
                                                                                  PrintBaz(value);
                                                                                  break;
                                                                              }
    }
}
```

```
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {













    }
}
```

```
void ProcessValue(int value)
{
    switch (value % 3)
    {
    case 0:
        PrintFoo(value);
        break;

    case 1:
        PrintBar(value);
        break;

    case 2:
        PrintBaz(value);
        break;
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        ProcessValue(value);
    }
}
```

```cpp
void ProcessValue(int value)
{
    switch (value % 3)
    {
    case 0:
        PrintFoo(value);
        break;

    case 1:
        PrintBar(value);
        break;

    case 2:
        PrintBaz(value);
        break;
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        ProcessValue(value);
    }
}
```

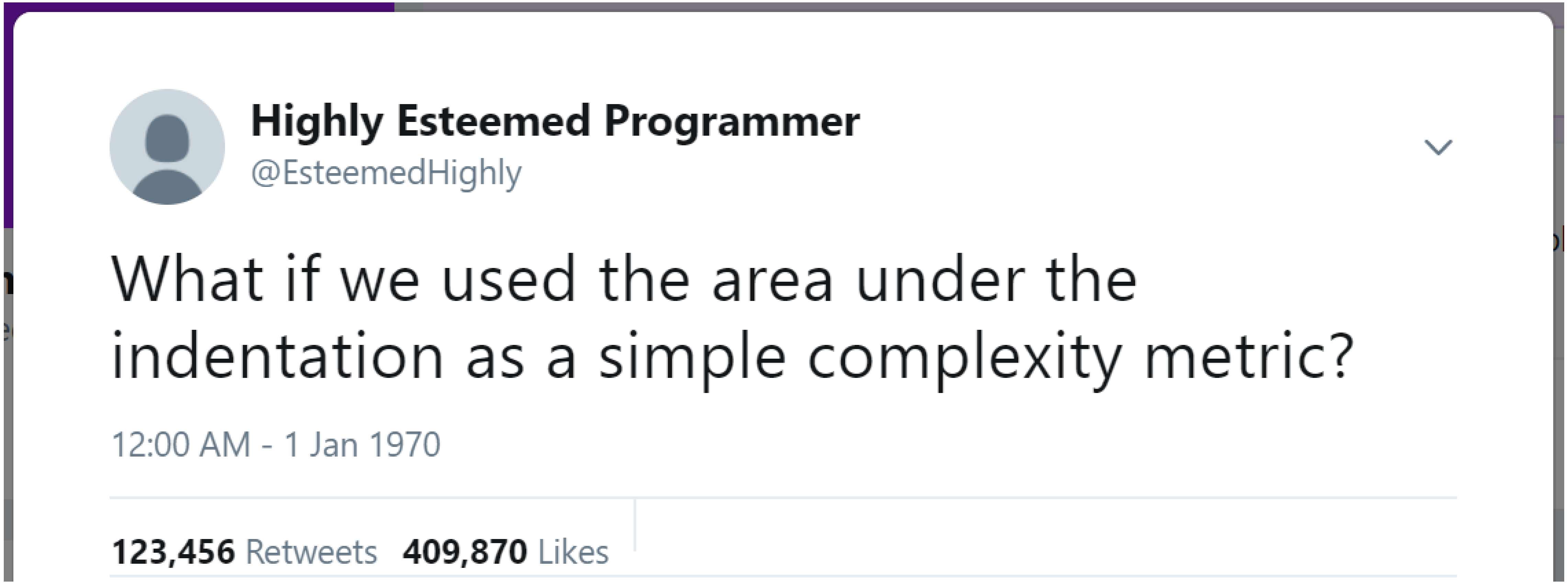
```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        ProcessValue(value);
    }
}
```

```cpp
void DoThing(int index)
{
    if (IsValidIndexOfOtherThing(index))
    {
        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
    }
}
```

```cpp
void DoThing(int index)
{
    if (!IsValidIndexOfOtherThing(index))
    {
        return;
    }

    if (!CanDoSomethingWithNumber(index))
    {
        return;
    }

    if (!CheckSomethingVeryCriticalAboutValue(index))
    {
        return;
    }

    for (auto const& value : GetValuesSimilarTo(index))
    {
        ProcessValue(value);
    }
}
```

Much Better

```cpp
void DoThing(int index)
{
    if (IsValidIndexOfOtherThing(index))
    {
        if (CanDoSomethingWithNumber(index))
        {
            if (CheckSomethingCriticalAboutValue(index))
            {
                for (auto const& value : GetData(index))
                {
                    switch (value % 3)
                    {
                    case 0:
                        PrintFoo(value);
                        break;

                    case 1:
                        PrintBar(value);
                        break;

                    case 2:
                        PrintBaz(value);
                        break;
                    }
                }
            }
        }
    }
}
```



Highly Esteemed Programmer
@EsteemedHighly

What if we used the area under the indentation as a simple complexity metric?

12:00 AM - 1 Jan 1970

123,456 Retweets **409,870** Likes

\0