Computer Programming II

Ming-Feng Tsai (Victor Tsai)

Dept. of Computer Science
National Chengchi University

Advanced Pointers

More about Pointers

- Example: foo_pointer.c

```
21 int main(){
22     int* a = NULL;
23     foo_1(a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- Example: foo_pointer.c

```
4  void foo_1(int* a, int size) {
5      a = (int*)calloc(size, sizeof(int));
6  }
```

```
21 int main(){
22     int* a = NULL;
23     foo_1(a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- Example: foo_pointer.c

```
4  void foo_1(int* a, int size) {
5      a = (int*)calloc(size, sizeof(int));
6  }
```

```
21 int main(){
22     int* a = NULL;
23     foo_1(a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- Example: foo_pointer.c

```
4  void foo_1(int* a, int size) {
5      a = (int*)calloc(size, sizeof(int));
6  }
```

```
21 int main(){
22     int* a = NULL;
23     foo_1(a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- Example: foo_pointer.c

```
4  void foo_1(int* a, int size) {
5      a = (int*)calloc(size, sizeof(int));
6  }
```

```
21 int main(){
22     int* a = NULL;
23     foo_1(a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

What is wrong with this code?

More about Pointers

- Fix the problem by returning a pointer

```
 8 int* foo_2(int size) {
 9     int* a;
10     a = (int*)calloc(size, sizeof(int));
11     return a;
12 }
```

More about Pointers

- Fix the problem by returning a pointer

```
 8 int* foo_2(int size) {
 9     int* a;
10     a = (int*)calloc(size, sizeof(int));
11     return a;
12 }
```

```
21 int main(){
22     int* a;
23     a = foo_2(3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- Fix the problem by returning a pointer

```
 8 int* foo_2(int size) {
 9     int* a;
10     a = (int*)calloc(size, sizeof(int));
11     return a;
12 }
```

```
21 int main(){
22     int* a;
23     a = foo_2(3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- Fix the problem by returning a pointer

```
 8 int* foo_2(int size) {
 9     int* a;
10     a = (int*)calloc(size, sizeof(int));
11     return a;
12 }
```

```
21 int main(){
22     int* a;
23     a = foo_2(3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- What if we don’t want to return a pointer

```
14 void foo_3(int **a, int size){
15     *a = (int*)calloc(size, sizeof(int));
16 }
```

More about Pointers

- What if we don’t want to return a pointer

```
14 void foo_3(int **a, int size){
15     *a = (int*)calloc(size, sizeof(int));
16 }
```

```
21 int main(){
22     int* a;
23     foo_3(&a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- What if we don’t want to return a pointer

```
14 void foo_3(int **a, int size){
15     *a = (int*)calloc(size, sizeof(int));
16 }
```

```
21 int main(){
22     int* a;
23     foo_3(&a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

- What if we don't want to return a pointer

```
14 void foo_3(int **a, int size){
15     *a = (int*)calloc(size, sizeof(int));
16 }
```

```
21 int main(){
22     int* a;
23     foo_3(&a, 3);
24     printf("%d\t%d\t%d\n", a[0], a[1], a[2]);
25     return 0;
26 }
```

More about Pointers

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

`int *a in main`

More about Pointers

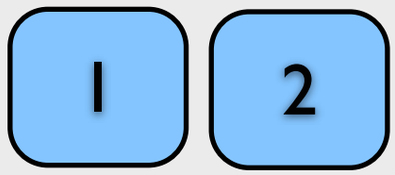
- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

`int *a in main`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`



`int **a in foo`

`int *a in main`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

`int *a in main`

More about Pointers

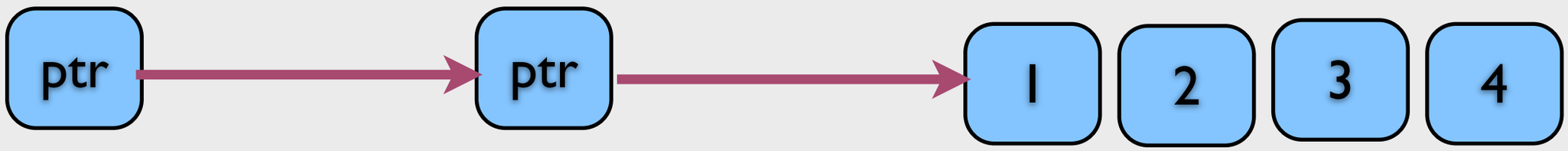
- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`

`int **a in foo`

`int *a in main`

More about Pointers

- If we don't want to return a pointer
  - Pass `&a` to main as a parameter
  - `void foo(int **a, int size) { ... }`



`int **a in foo`

`int *a in main`

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

More about Pointer

	Address	Content
'*a' in main function	0x0224	0x0234
	0x0228	
	0x022C	
'**a' in foo function	0x0230	0x0224
	0x0234	0
	0x0238	1
	0x023C	2

- **`main()`**: pass **`&a (0x224)`** to **`**a`** in **`foo()`**
- **`foo()`**: **`*a = malloc(...)`** // suppose malloc return 0x234
- **`foo()`**: **`(*a)[i] = i`**

More about Pointer

- Difference between this and swap
  - Comparison

- Actually, we copy the address of the pointer to **`int`** from main to **`foo`**. So, if we dereference **`'a'`** in **`foo`**, it will become the pointer to int in main function

More about Pointer

- Difference between this and swap
  - Comparison

```
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

- Actually, we copy the address of the pointer to **`int`** from main to **`foo`**. So, if we dereference **`'a'`** in **`foo`**, it will become the pointer to int in main function

More about Pointer

- Difference between this and swap
  - Comparison

```
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

	type	dereference
int **a	pointer to pointer to int	pointer to int
int *a	pointer to int	int

- Actually, we copy the address of the pointer to **`int`** from main to **`foo`**. So, if we dereference **`'a'`** in **`foo`**, it will become the pointer to int in main function

Function Pointer

- Declare a function

Function Pointer

- Declare a function

```
int func (int a, float b); /* function returning an int */
```

Function Pointer

- Declare a function

```
int func (int a, float b); /* function returning an int */

int *func (int a, float b); /* function returning pointer to int */
```

Function Pointer

- Declare a function

```
int func (int a, float b); /* function returning an int */

int *func (int a, float b); /* function returning pointer to int */

int (*func) (int a, float b); /* pointer to function returning int*/
```

Function Pointer

- Declare a function

```
int func (int a, float b); /* function returning an int */

int *func (int a, float b); /* function returning pointer to int */

int (*func) (int a, float b); /* pointer to function returning int*/
```

- Function Pointer
- Once you've got the pointer, you can assign the address of the right sort of function just by using its name
- Call the function using one of two forms
  - `(*func) (1, 2)`
  - `func(1, 2)`

Function Pointer

- Example: fun_ptr.c

```c
int sum(int x, int y) {
    return x + y;
}


int product(int x, int y) {
    return x * y;
}


int difference(int x, int y) {
    return x - y;
}
```

```c
    int (*pfun)(int, int); /* Function pointer declaration */

    pfun = sum;
    result = pfun(a, b); /* Call sum() through pointer */
    printf("\npfun = sum result = %d", result);

    pfun = product;
    result = pfun(a, b); /* Call product() through pointer */
    printf("\npfun = product result = %d", result);

    pfun = difference;
    result = pfun(a, b); /* Call difference() through pointer */
    printf("\npfun = difference result = %d\n", result);
```

Function Pointer

- Example: fun_ptr.c

```c
int sum(int x, int y) {
    return x + y;
}


int product(int x, int y) {
    return x * y;
}


int difference(int x, int y) {
    return x - y;
}
```

```c
    int (*pfun)(int, int); /* Function pointer declaration */

    pfun = sum;
    result = pfun(a, b); /* Call sum() through pointer */
    printf("\npfun = sum result = %d", result);

    pfun = product;
    result = pfun(a, b); /* Call product() through pointer */
    printf("\npfun = product result = %d", result);

    pfun = difference;
    result = pfun(a, b); /* Call difference() through pointer */
    printf("\npfun = difference result = %d\n", result);
```

Function Pointer

- Example: fun_ptr.c

```c
int sum(int x, int y) {
    return x + y;
}


int product(int x, int y) {
    return x * y;
}


int difference(int x, int y) {
    return x - y;
}
```

```c
    int (*pfun)(int, int); /* Function pointer declaration */

    pfun = sum;
    result = pfun(a, b); /* Call sum() through pointer */
    printf("\npfun = sum result = %d", result);

    pfun = product;
    result = pfun(a, b); /* Call product() through pointer */
    printf("\npfun = product result = %d", result);

    pfun = difference;
    result = pfun(a, b); /* Call difference() through pointer */
    printf("\npfun = difference result = %d\n", result);
```

Function Pointer

- Example: fun_ptr.c

```c
int sum(int x, int y) {
    return x + y;
}


int product(int x, int y) {
    return x * y;
}


int difference(int x, int y) {
    return x - y;
}
```

```c
    int (*pfun)(int, int); /* Function pointer declaration */

    pfun = sum;
    result = pfun(a, b); /* Call sum() through pointer */
    printf("\npfun = sum result = %d", result);

    pfun = product;
    result = pfun(a, b); /* Call product() through pointer */
    printf("\npfun = product result = %d", result);

    pfun = difference;
    result = pfun(a, b); /* Call difference() through pointer */
    printf("\npfun = difference result = %d\n", result);
```

Function Pointer

- Example: fun_ptr.c

```c
int sum(int x, int y) {
    return x + y;
}


int product(int x, int y) {
    return x * y;
}


int difference(int x, int y) {
    return x - y;
}
```

```c
    int (*pfun)(int, int); /* Function pointer declaration */

    pfun = sum;
    result = pfun(a, b); /* Call sum() through pointer */
    printf("\npfun = sum result = %d", result);

    pfun = product;
    result = pfun(a, b); /* Call product() through pointer */
    printf("\npfun = product result = %d", result);

    pfun = difference;
    result = pfun(a, b); /* Call difference() through pointer */
    printf("\npfun = difference result = %d\n", result);
```

Function Pointer

- Example: fun_ptr_ary.c

```c
int sum(int x, int y){
    return x + y;
}

int product(int x, int y){
    return x * y;
}

int difference(int x, int y){
    return x - y;
}
```

```c
    int (*pfun[3])(int, int); /* Function pointer array declaration */

    /* Initialize pointers */
    pfun[0] = sum;
    pfun[1] = product;
    pfun[2] = difference;

    int i;
    for(i = 0 ; i < 3 ; i++)
    {
        result = pfun[i](a, b); /* Call the function through a pointer */
        printf("result = %d\n", result);  /* Display the result */
    }
```

Function Pointer

- Example: fun_ptr_ary.c

```c
int sum(int x, int y){
    return x + y;
}

int product(int x, int y){
    return x * y;
}

int difference(int x, int y){
    return x - y;
}
```

```c
    int (*pfun[3])(int, int); /* Function pointer array declaration */

    /* Initialize pointers */
    pfun[0] = sum;
    pfun[1] = product;
    pfun[2] = difference;

    int i;
    for(i = 0 ; i < 3 ; i++)
    {
        result = pfun[i](a, b); /* Call the function through a pointer */
        printf("result = %d\n", result);  /* Display the result */
    }
```

Function Pointer

- Example: fun_ptr_ary.c

```c
int sum(int x, int y){
    return x + y;
}

int product(int x, int y){
    return x * y;
}

int difference(int x, int y){
    return x - y;
}
```

```c
    int (*pfun[3])(int, int); /* Function pointer array declaration */

    /* Initialize pointers */
    pfun[0] = sum;
    pfun[1] = product;
    pfun[2] = difference;

    int i;
    for(i = 0 ; i < 3 ; i++)
    {
        result = pfun[i](a, b); /* Call the function through a pointer */
        printf("result = %d\n", result);  /* Display the result */
    }
```

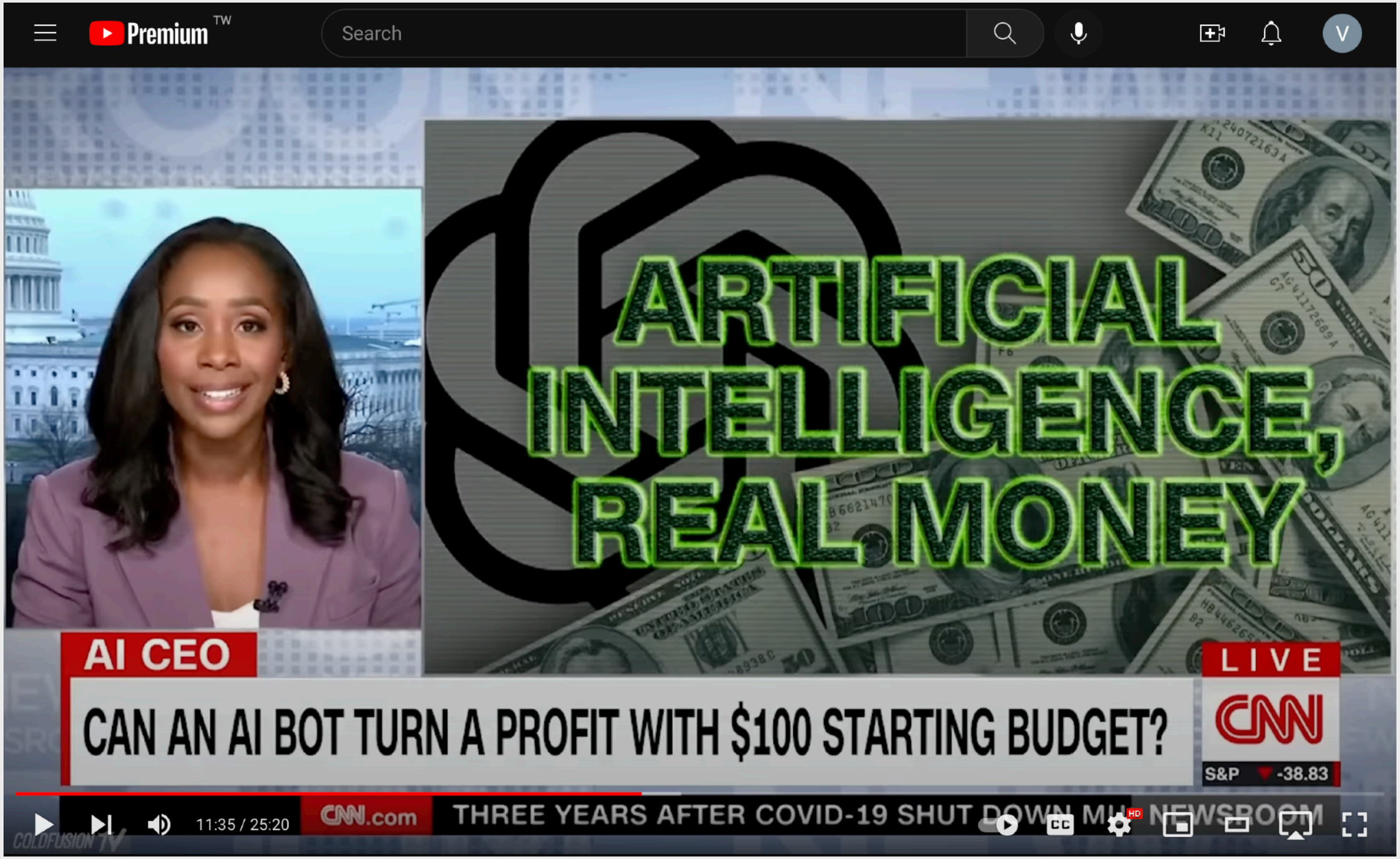
Function Pointer

- Example: fun_ptr_ary.c

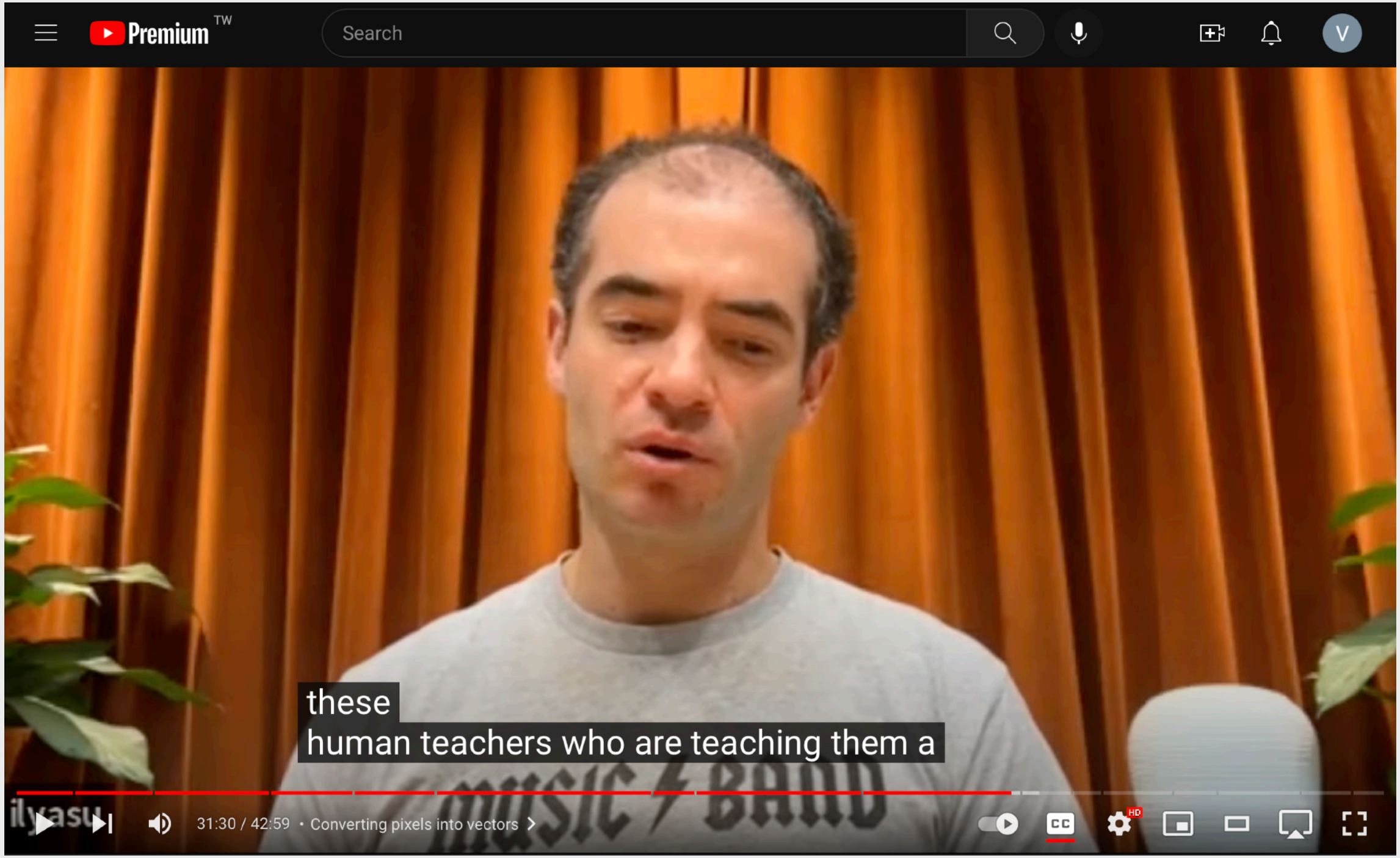
```c
int sum(int x, int y){
    return x + y;
}

int product(int x, int y){
    return x * y;
}

int difference(int x, int y){
    return x - y;
}
```

```c
    int (*pfun[3])(int, int); /* Function pointer array declaration */

    /* Initialize pointers */
    pfun[0] = sum;
    pfun[1] = product;
    pfun[2] = difference;

    int i;
    for(i = 0 ; i < 3 ; i++)
    {
        result = pfun[i](a, b); /* Call the function through a pointer */
        printf("result = %d\n", result);  /* Display the result */
    }
```

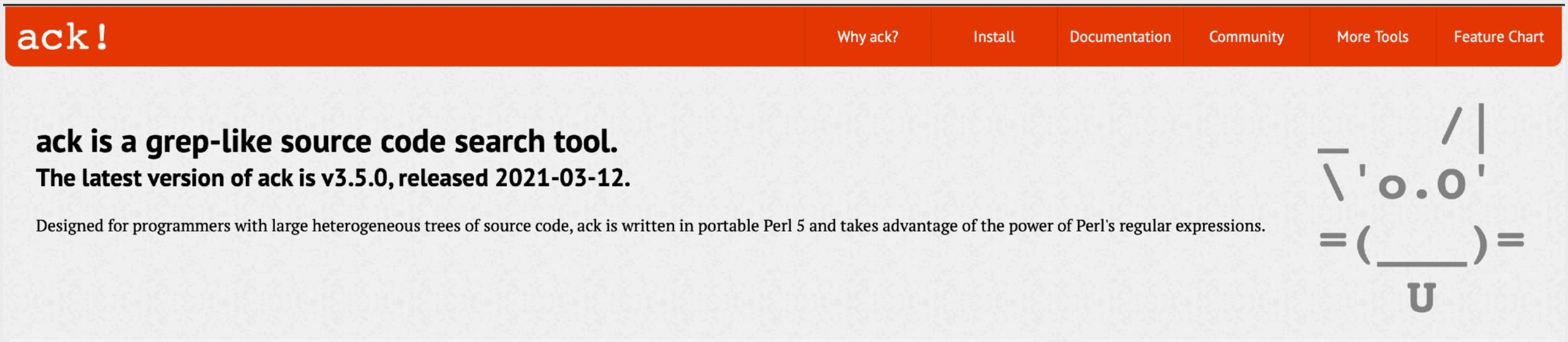
AI is Evolving Faster Than You Think [GPT-4 and beyond]



GPT-4 Creator Ilya Sutskever



Unix Tips

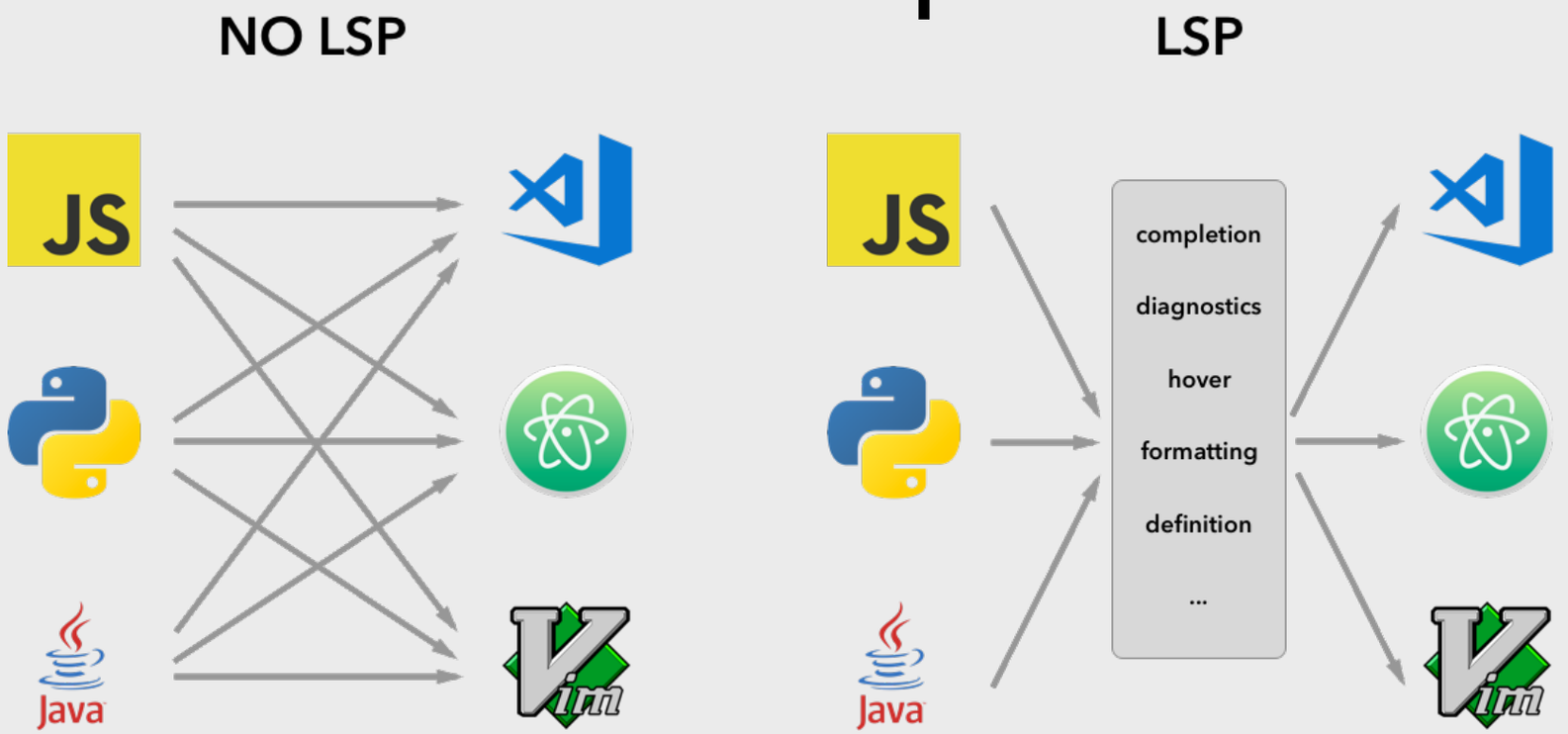


- Ack is a grep-like source code search tool
  - https://beyondgrep.com
  - It is designed for programmers with large heterogeneous trees of source code, ack is written in portable Perl 5 and takes advantage of the power of Perl's regular expressions.

Unix Tips

- Ag: A code searching tool similar to ack, with a focus on speed.
  - https://github.com/ggreer/the_silver_searcher
  - Ack and Ag found the same results, but Ag was 34x faster.

Unix Tips



- Language Server Protocol
  - The Language Server Protocol (LSP) defines the protocol used between an editor or IDE and a language server that provides language features like auto complete, go to definition, find all references etc. The goal of the Language Server Index Format (LSIF, pronounced like "else if") is to support rich code navigation in development tools or a Web UI without needing a local copy of the source code.