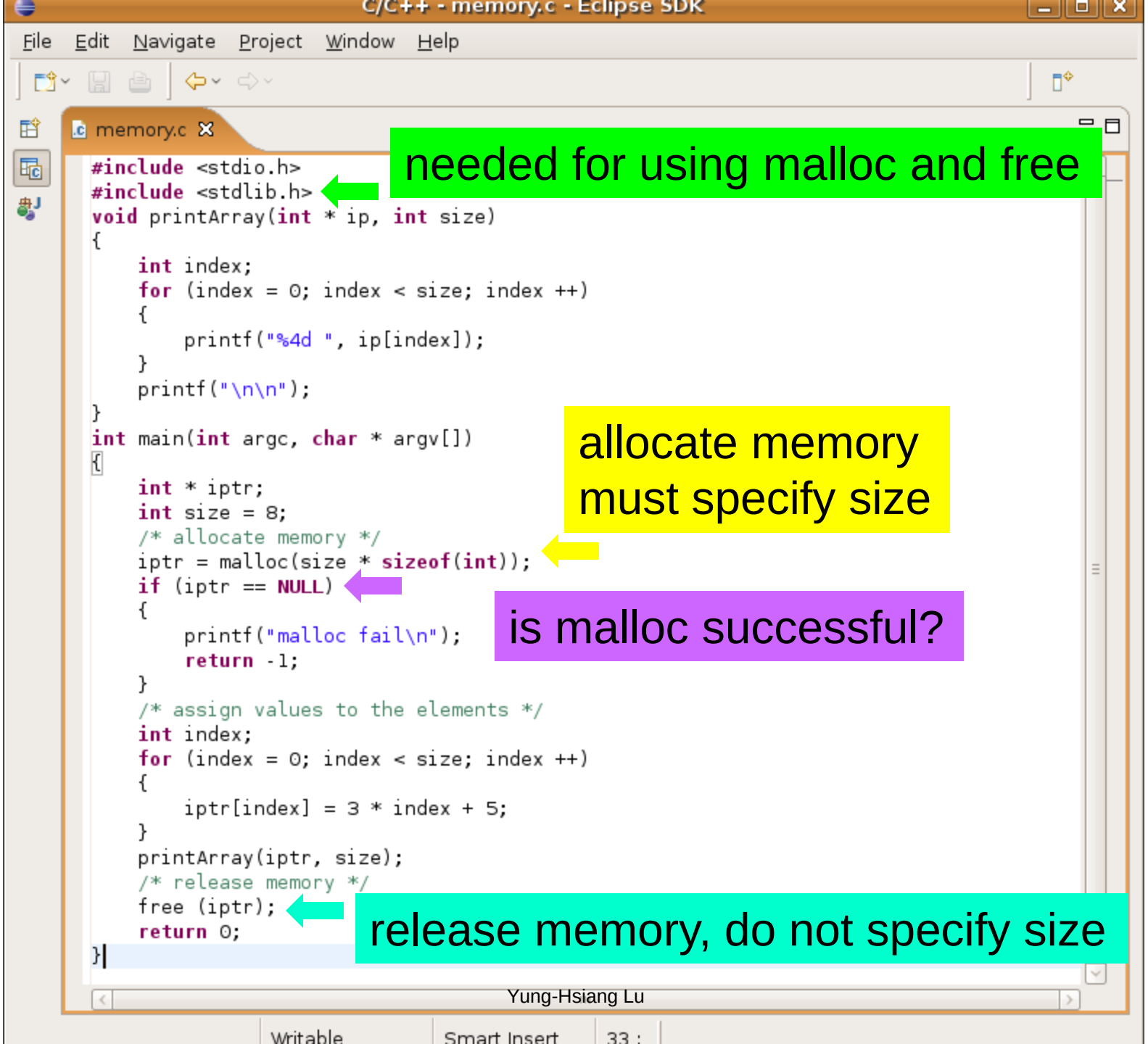
Allocate and Release Memory

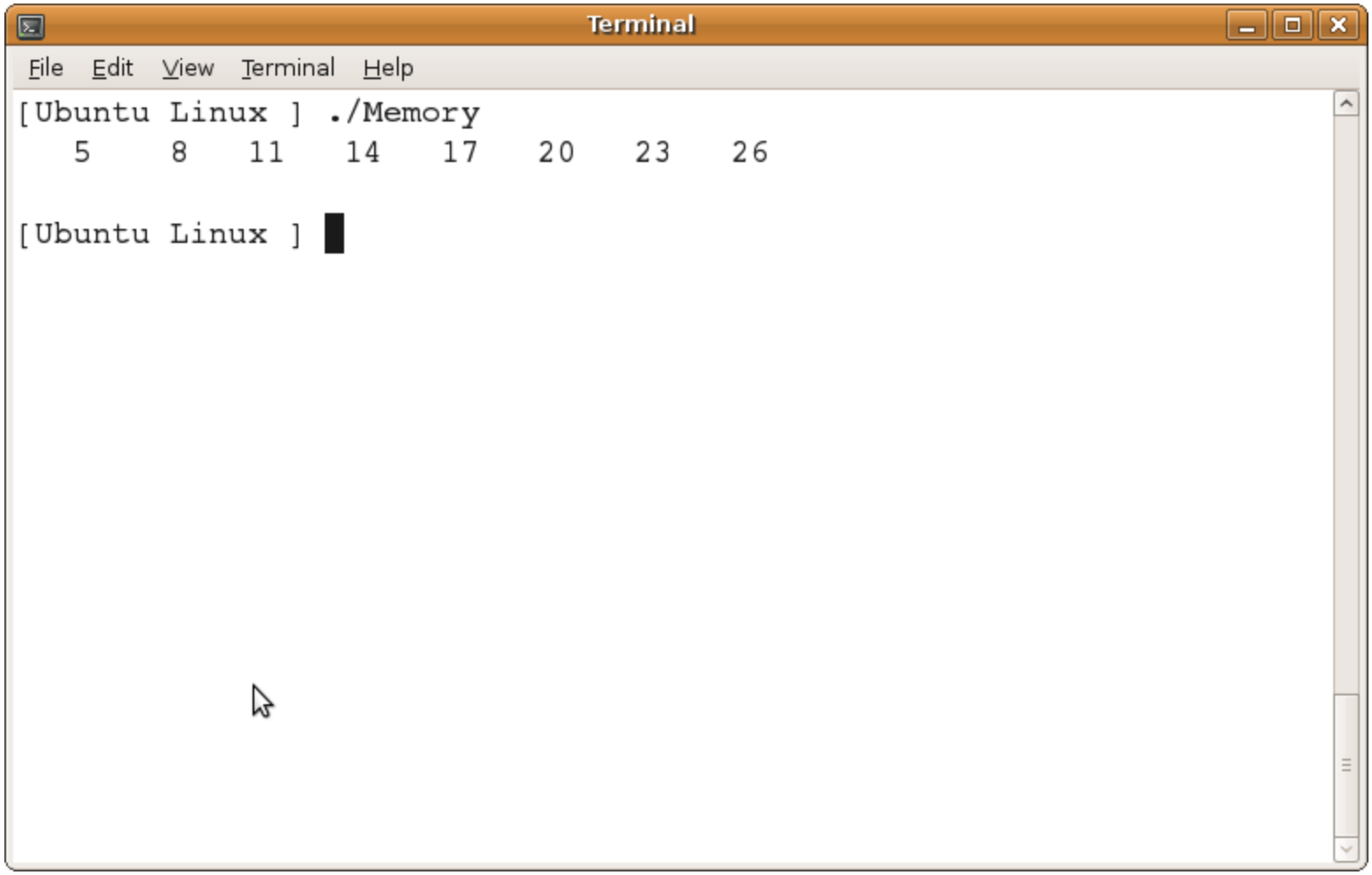
Yung-Hsiang Lu

Why to Allocate and Release?

- Memory stores data. In many scenarios, programmers do not know how much memory is needed when writing the programs.
- If more memory is needed, a program must be able to **allocate** more memory.
- If **allocated** memory is no longer needed, the memory should be **released**.
- If memory is not **explicitly** allocated by the program, the memory cannot be explicitly released by the program.


C/C++ - memory.c - Eclipse SDK
File Edit Navigate Project Window Help
memory.c
#include <stdio.h>
#include <stdlib.h>
void printArray(int * ip, int size)
{
int index;
for (index = 0; index < size; index ++)
{
printf("%4d ", ip[index]);
}
printf("\n\n");
}
int main(int argc, char * argv[])
{
int * iptr;
int size = 8;
/* allocate memory */
iptr = malloc(size * sizeof(int));
if (iptr == NULL)
{
printf("malloc fail\n");
return -1;
}
/* assign values to the elements */
int index;
for (index = 0; index < size; index ++)
{
iptr[index] = 3 * index + 5;
}
printArray(iptr, size);
/* release memory */
free (iptr);
return 0;
}
needed for using malloc and free
allocate memory
must specify size
is malloc successful?
release memory, do not specify size
Writable
Smart Insert
33 :

Program Output



DESCRIPTION

The **malloc**(), **calloc**(), **valloc**(), **realloc**(), and **reallocf**() functions allocate memory. The allocated memory is aligned such that it can be used for any data type, including AltiVec- and SSE-related types. The **free**() function frees allocations that were created via the preceding allocation functions.

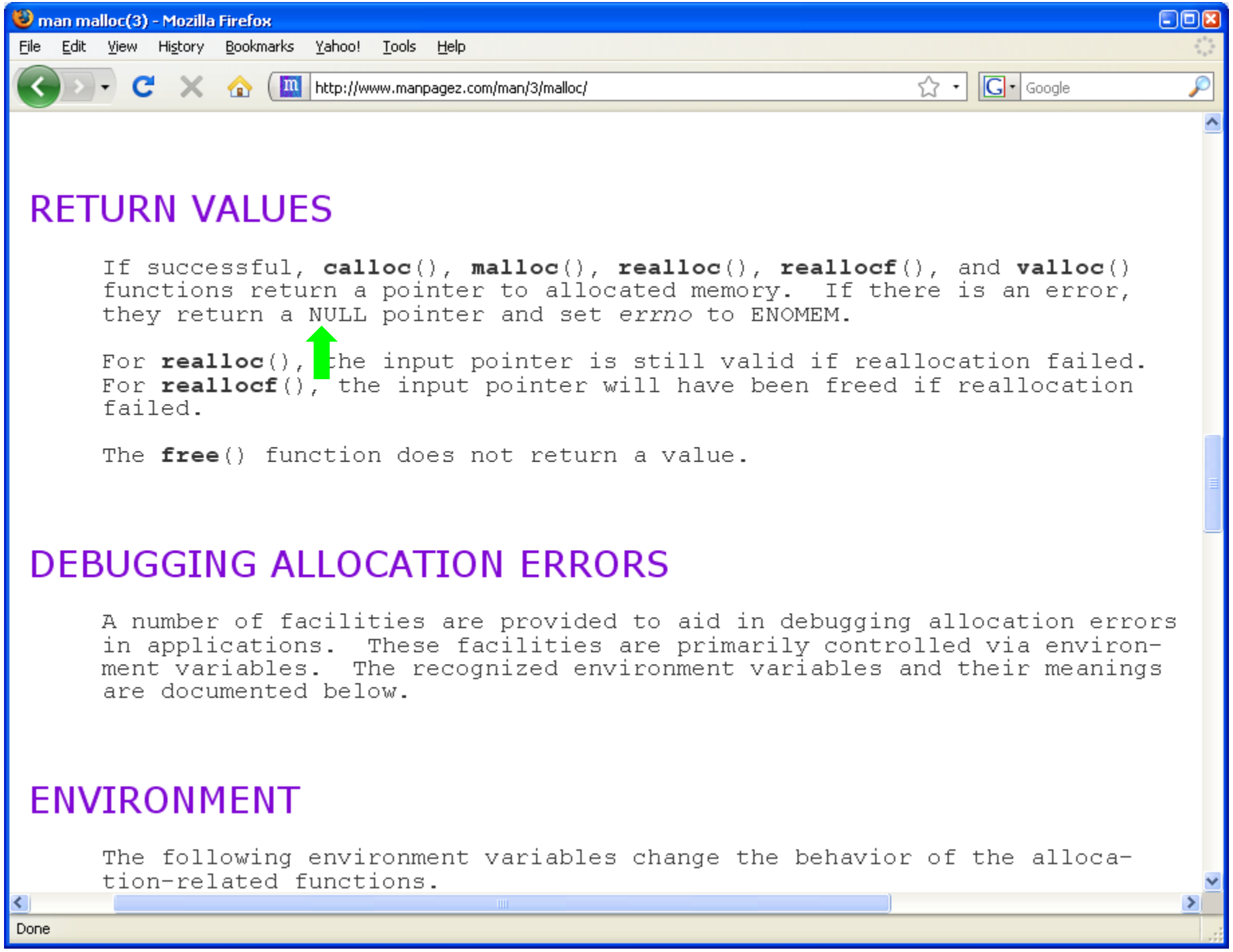
The **malloc**() function allocates *size* bytes of memory and returns a pointer to the allocated memory.

The **calloc**() function contiguously allocates enough space for *count* objects that are *size* bytes of memory each and returns a pointer to the allocated memory. The allocated memory is filled with bytes of value zero.

The **valloc**() function allocates *size* bytes of memory and returns a pointer to the allocated memory. The allocated memory is aligned on a page boundary.

The **realloc**() function tries to change the size of the allocation pointed to by *ptr* to *size*, and returns *ptr*. If there is not enough room to enlarge the memory allocation pointed to by *ptr*, **realloc**() creates a new allocation, copies as much of the old data pointed to by *ptr* as will fit to the new allocation, frees the old allocation, and returns a pointer to the allocated memory. If *ptr* is NULL, **realloc**() is identical to a call to **malloc**() for *size* bytes. If *size* is zero and *ptr* is not NULL, a new, minimum sized object is allocated and the original object is freed.

The **reallocf**() function is identical to the **realloc**() function, except that it will free the passed pointer when the requested memory cannot be allocated. This is a FreeBSD specific API designed to ease the problems with traditional coding styles for reallocf causing memory leaks in

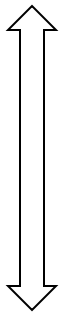

man malloc(3) - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://www.manpagez.com/man/3/malloc/
Google
RETURN VALUES
If successful, calloc(), malloc(), realloc(), reallocf(), and valloc() functions return a pointer to allocated memory. If there is an error, they return a NULL pointer and set errno to ENOMEM.
For realloc(), the input pointer is still valid if reallocation failed. For reallocf(), the input pointer will have been freed if reallocation failed.
The free() function does not return a value.
DEBUGGING ALLOCATION ERRORS
A number of facilities are provided to aid in debugging allocation errors in applications. These facilities are primarily controlled via environment variables. The recognized environment variables and their meanings are documented below.
ENVIRONMENT
The following environment variables change the behavior of the allocation-related functions.
Done

How Much Memory Is Used?

Assume we are using a 32-bit (4 byte) computer and each integer occupies 4 bytes.

```
int * ip;                                  /* 4 bytes */
ip = malloc (16 * sizeof(int));            /* 64 bytes */
                                           /* total 68 bytes */
```

address	value
somewhere (& ip)	XXXXX
somewhere XXXXX	ip[0]
XXXXX + 3	ip[1]
XXXXX + 7	ip[2]
….	…



64 bytes

Memory Management

- **Always** check whether malloc is successful before using the memory.
- Always release memory if it is no longer needed.
- Do not release memory if it may be used later.
- C does **not** check whether array indexes are within the allowed range. It is the programmer's responsibility.

```
int * ip = malloc(16 * sizeof(int));
ip[100] = 9876;   /* gcc does not report any problem */
/* The program will crash during execution */
```

- Do not give zero or negative numbers to malloc.

Two-Dimensional Array (Matrix)

column

Flight Time	Boston	Chicago	New York	San Francisco
Boston		150	60	270
Chicago	140		130	240
New York	60	140		290
San Francisco	260	230	280	

row

The flight time may not be symmetric due to wind.

High-Dimensional Arrays

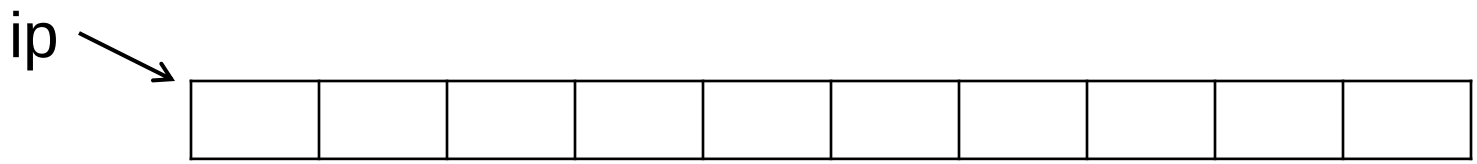
```
int a[20];              /* one-dimensional, 20 elements */
int b[10][6];           /* 60 elements */
double c[5][3][2];      /* 30 = 5   3    2 elements */

b[1][3] = 8;
a[3] = b[5][4];
c[2][1][0] = 53.987;
```

Allocated Memory = One-Dimensional Array

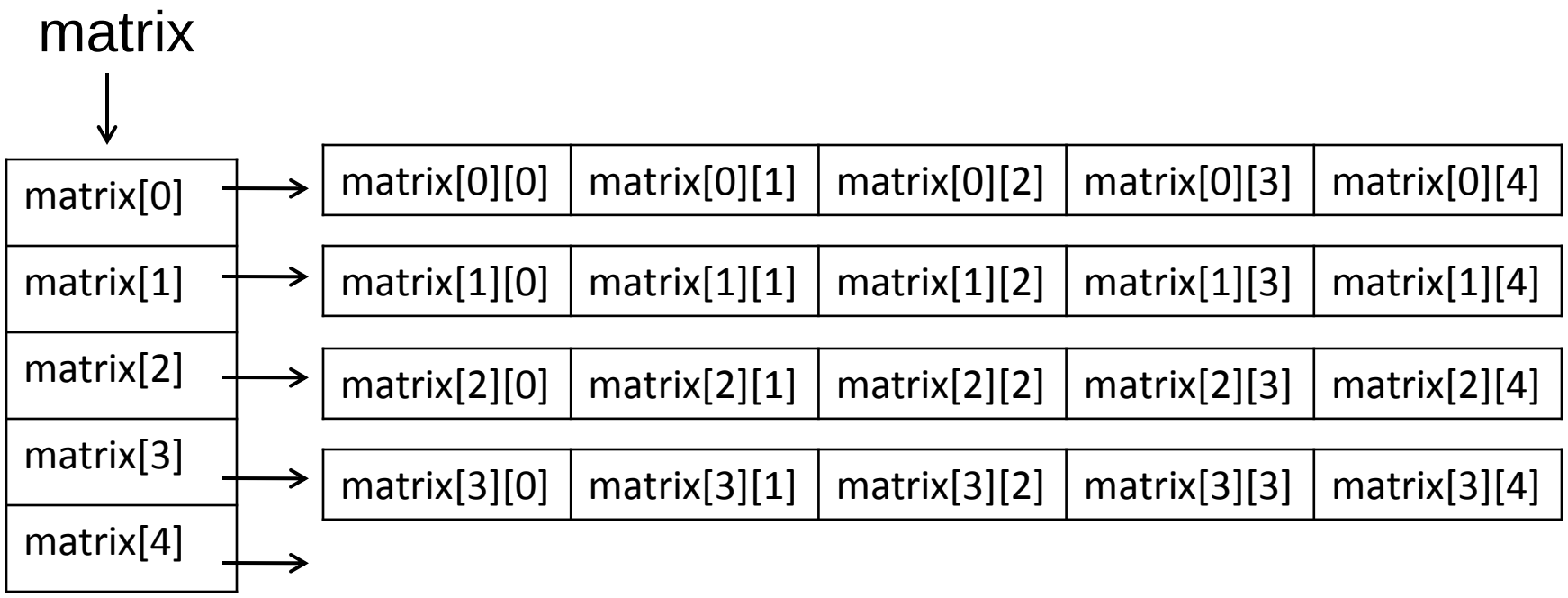
```
int * ip = malloc(10 * sizeof(int));

    /* ip points to a pieces of memory for 10 integers */
```



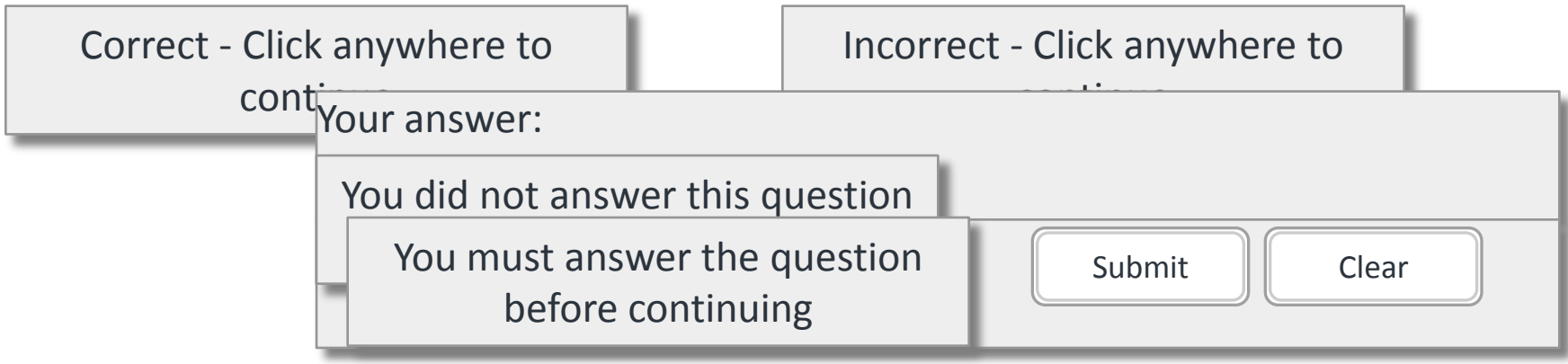
2-D Array = Pointers to Pointers

```
int * * matrix = malloc(numRow * sizeof(int *));
for (row = 0; row < numRow; row ++)
{   matrix [row] = malloc(numCol * sizeof(int));    }
```



Which statement is correct? You can choose multiple anssers.

- [x] A) malloc has one argument and it is the number of bytes to allocate
- [ ] B) free has one argument and it is the number of bytes to release
- [x] C) If malloc fails, the return value is NULL.
- [ ] D) The data stored in the memory allocated by malloc are reset to zero.



Allocate and Release

Your Score	{score}
Max Score	{max-score}
Number of Quiz Attempts	{total-attempts}

Question Feedback/Review Information Will Appear Here

Continue | Review Quiz