

# Arrays

Song Liu ([song.liu@bristol.ac.uk](mailto:song.liu@bristol.ac.uk))

GA 18, Fry Building,

Microsoft Teams (search "song liu").

# Previously

- Time Complexity
- Recursion
- Stack Memory Layout

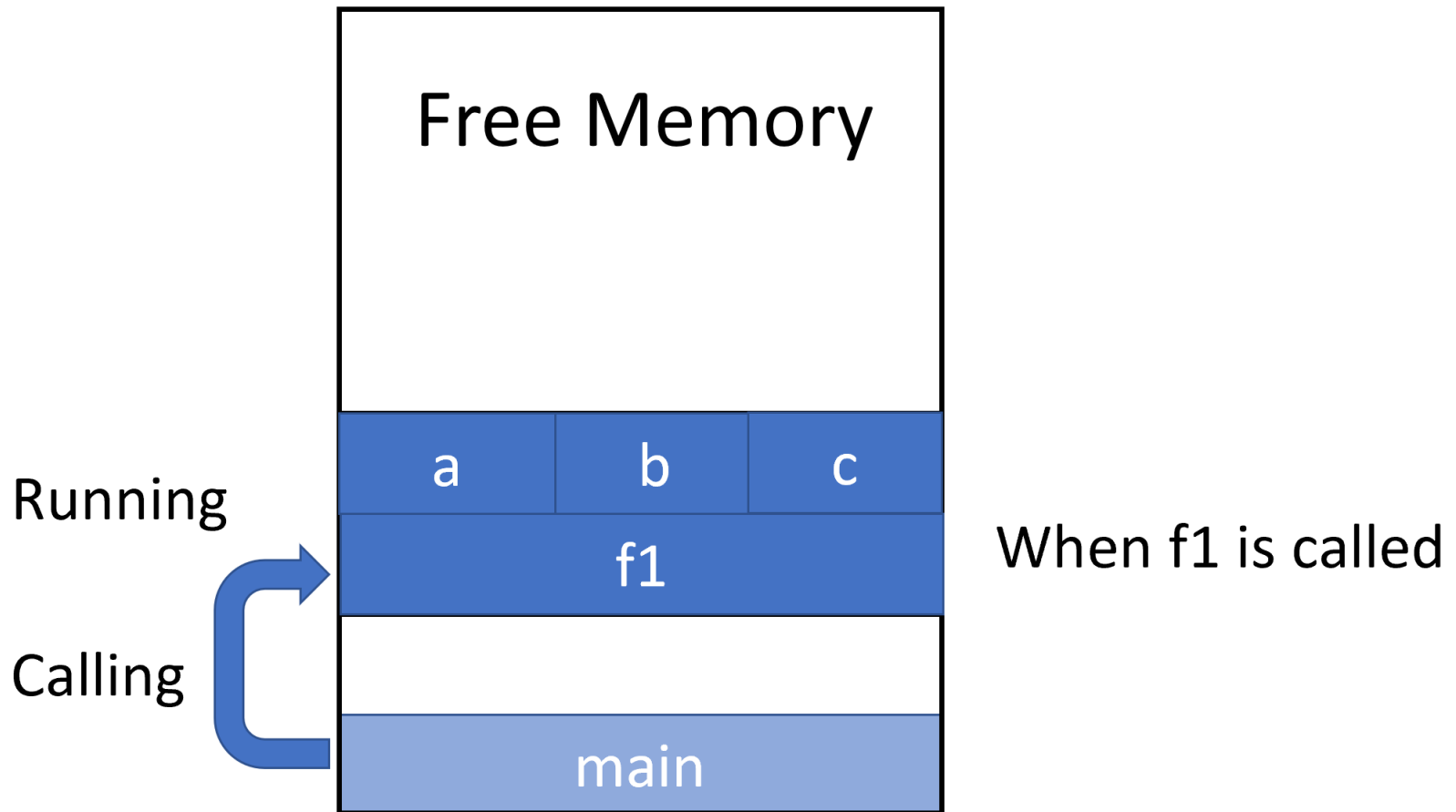
# Stack Memory Allocation

Consider a situation where function `f1` calls `f2` and `f2` calls `f3` .

```
#include <stdio.h>
void f3(){
    int d = 6, e = 7;
};
void f2(){
    int a = 4, b = 5;
    f3();
}
void f1(){
    int a = 1, b = 2, c = 3;
    f2();
};
void main(){
    f1();
}
```

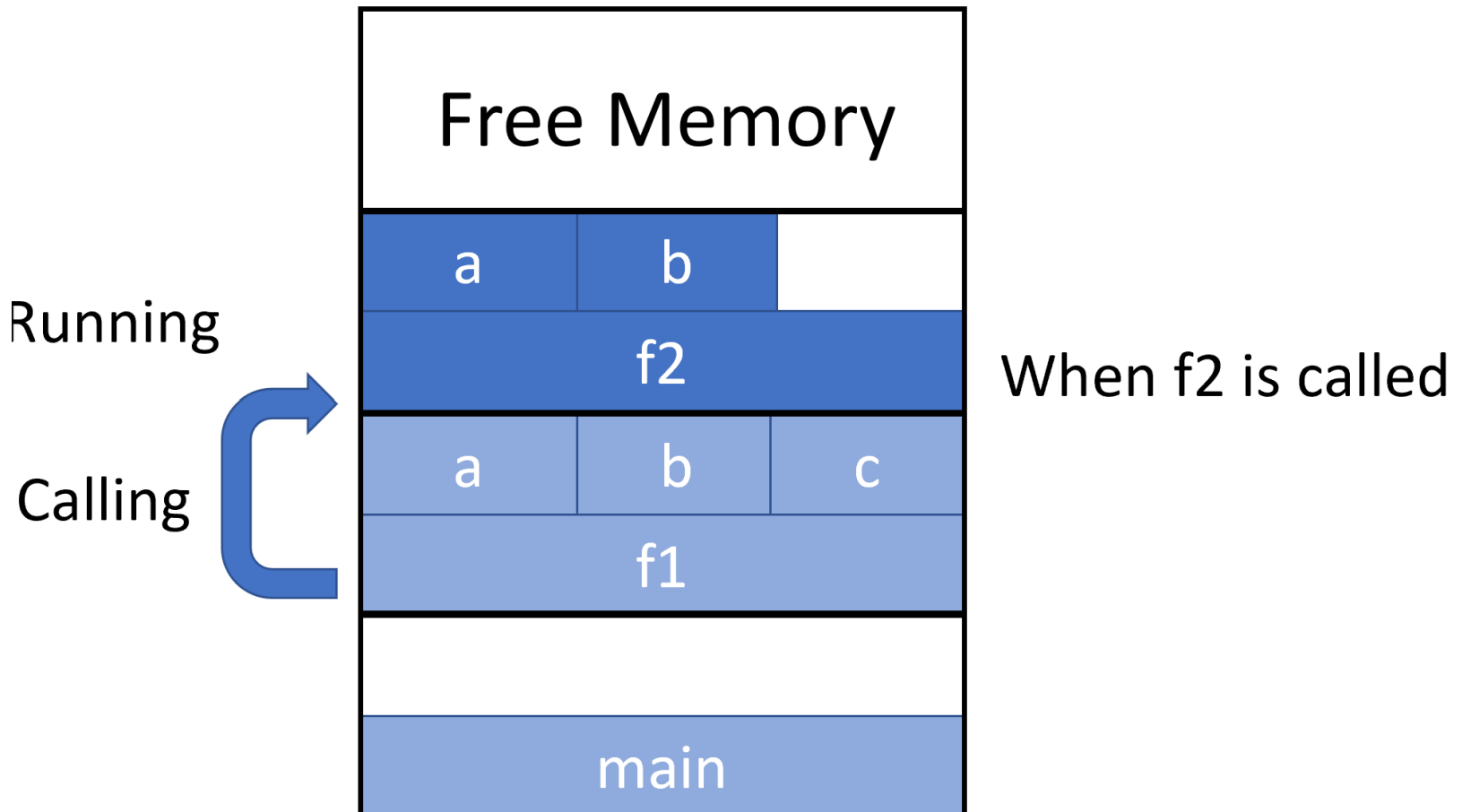
# Stack Memory Allocation

Below is the stack while `f1` is called.



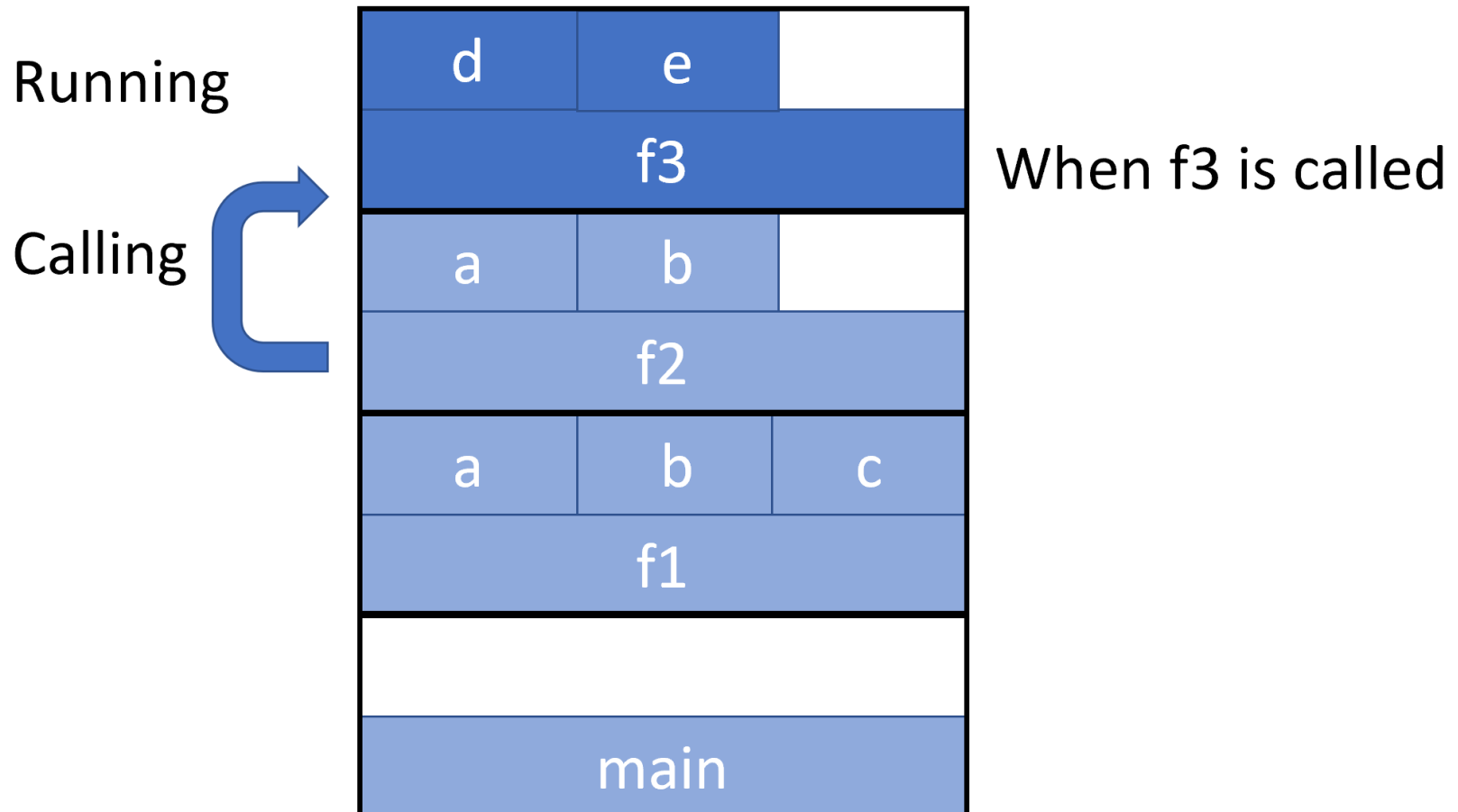
# Stack Memory Allocation

Below is the stack while `f2` is called.



# Stack Memory Allocation

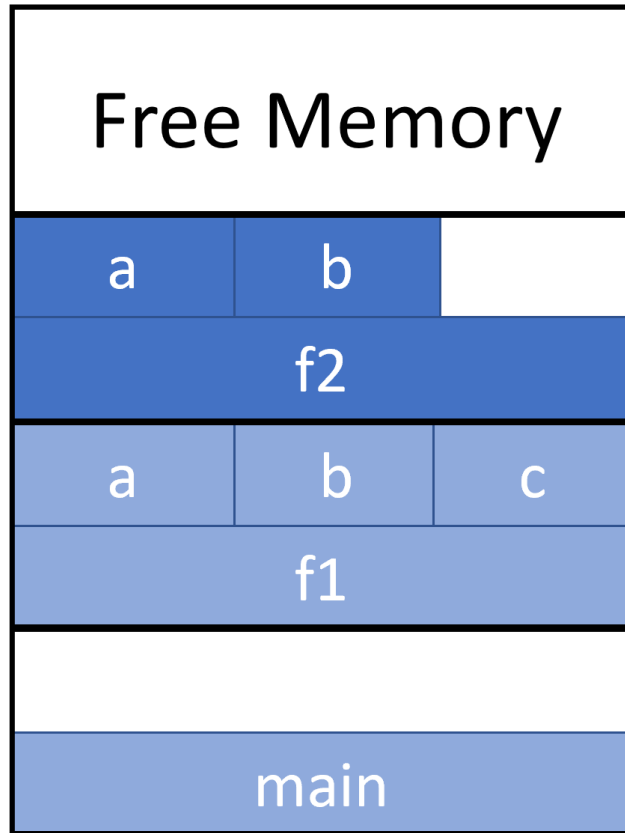
Below is the stack while `f3` is called.



# Stack Memory Release

Below is the stack while `f3` is finished.

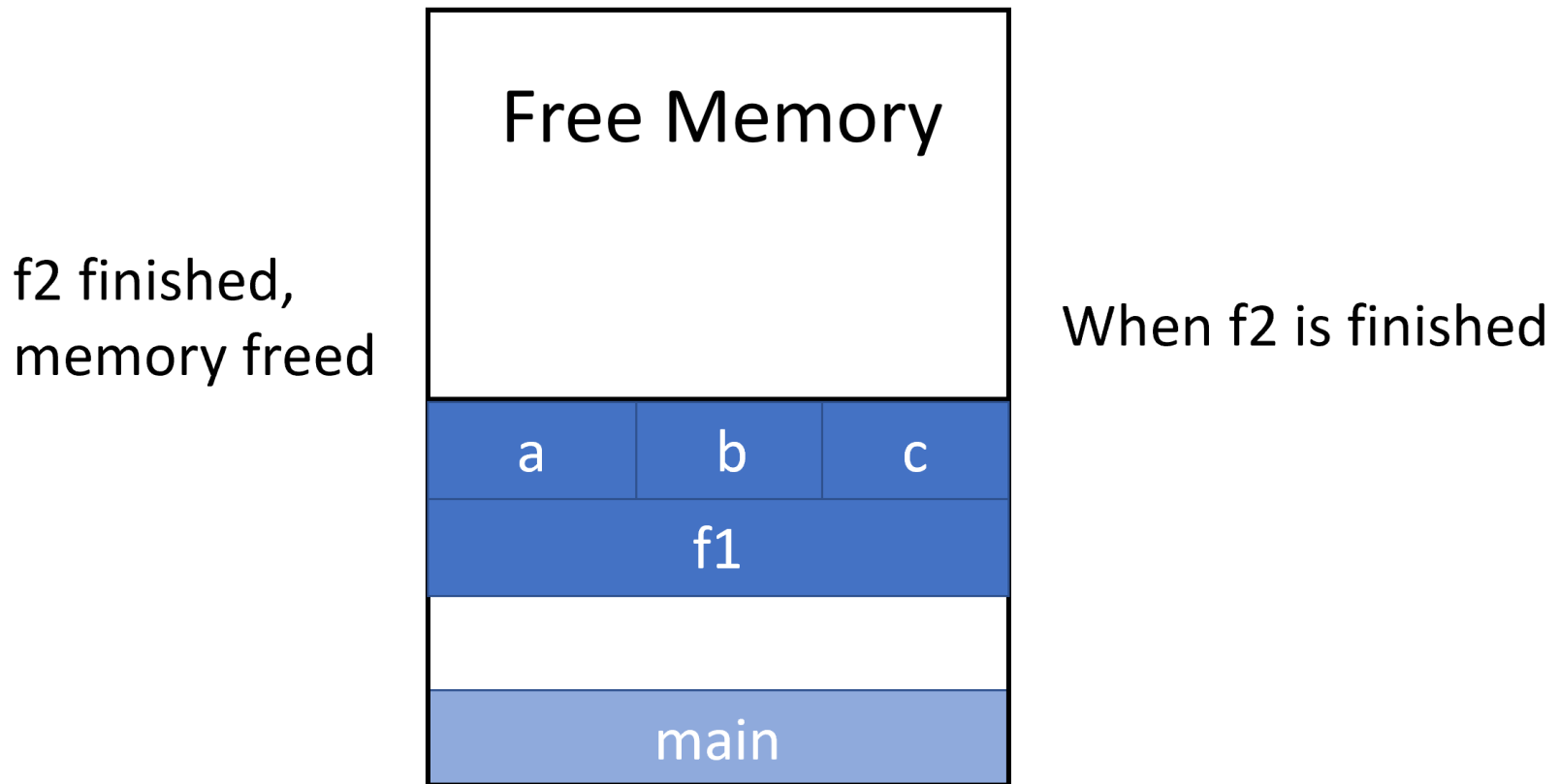
f3 finished,  
memory freed



When f3 is finished

# Stack Memory Release

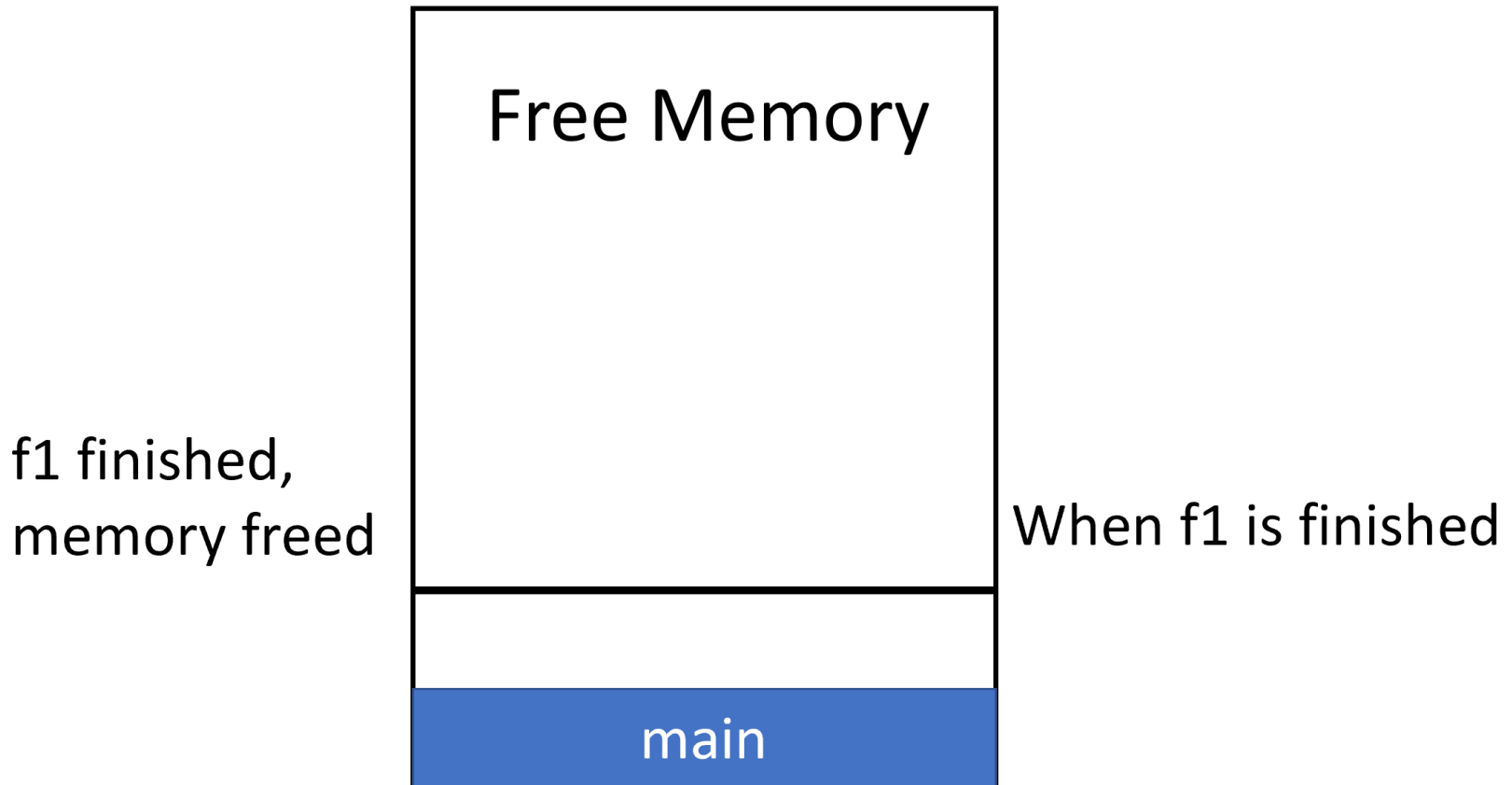
Below is the stack while `f2` is finished.





# Stack Memory Release

Below is the stack while `f1` is finished.



## Part II Data Science in C

- We have finished the first part of this course, the foundation of C.
- From this lecture, we will progress to the next section, which is using C to perform some basic DS tasks.
- First, we will look at how various types of data are stored in the memory.

# Today's Agenda

- **What is Array?**
  - How to
    - create an array?
    - access an array?
  - How an array is stored in the computer memory?
- How to pass an array as an input argument of a function?
  - Passing by value vs Passing by reference

# Vector Calculation

- Vectors are a sequence of numbers.
  - $\mathbf{a} = [1, 2, 3]$  is a three-dimensional vector.
- Vector calculations: addition, subtraction, dot product, etc.

$$\begin{array}{c} + \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 3 & 4 \\ \hline \end{array} \\ \\ = \\ \begin{array}{|c|c|c|} \hline 3 & 5 & 7 \\ \hline \end{array} \end{array}$$

Vector Addition

$$\begin{array}{c} \cdot \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 2 & 3 & 4 \\ \hline \end{array} \\ \\ = \\ \begin{array}{|c|} \hline 20 \\ \hline \end{array} \end{array}$$

Dot Product

# Vector Calculation

- Consider the following program tries to compute the three dimensional vector addition
  - $c = a + b, a = [1, 2, 3], b = [2, 3, 4]$ .

```
#include <stdio.h>
void main(){
    double a1=1.0, a2=2.0, a3=3.0;
    double b1=2.0, b2=3.0, b3=4.0;
    double c1, c2, c3;
    c1 = a1+b1; c2 = a2 + b2; c3= a3 + b3;
    ...
}
```

- This program is dumb.
- What if you have a 100-dimension vector?
  - `c1 = a1 + b1; ... c100 = a100 + b100; ?`

# Vector Calculation

To automate vector calculations, we need two things:

- A way to access an element in vector via an integer index.
  - e.g. `a[i]` represents the `i`-th element of vector `a`.
- A loop to perform calculations on all elements in a vector

Imagine we can do something like:

(pseudo code)

```
For i from 1 to 100  
    c[i] = a[i] + b[i]
```

This program is much more scalable and understandable.

# Array

- Array is a fundamental **data structure** in C programming language that **stores a sequence of elements**.
- You can declare an array using the syntax:
  - `data_type variable_name[array_size];` .
  - ```
// declares an int array with 100 elements.  
int a[100];
```
- `array_size` can **NOT** be a variable.
  - ```
int c = 100;  
int b[c]; // compilation error!
```
  - `array_size` must be determined at the time of compilation.

# Accessing Array

- The first element in the array is `a[0]` .
- The second element in the array is `a[1]` .
- and so on.
- e.g., `a[2] = 5;` assigns 5 to the third element of `a` .
- You can access multiple elements using a loop:

```
int a[10];
for(int i = 0; i < 10; i = i+1){
    a[i] = 123; // assigning the i+1-th element
}
for(int i = 0; i < 10; i = i+1){
    printf("%d ", a[i]); // print the i+1-th element
}
```

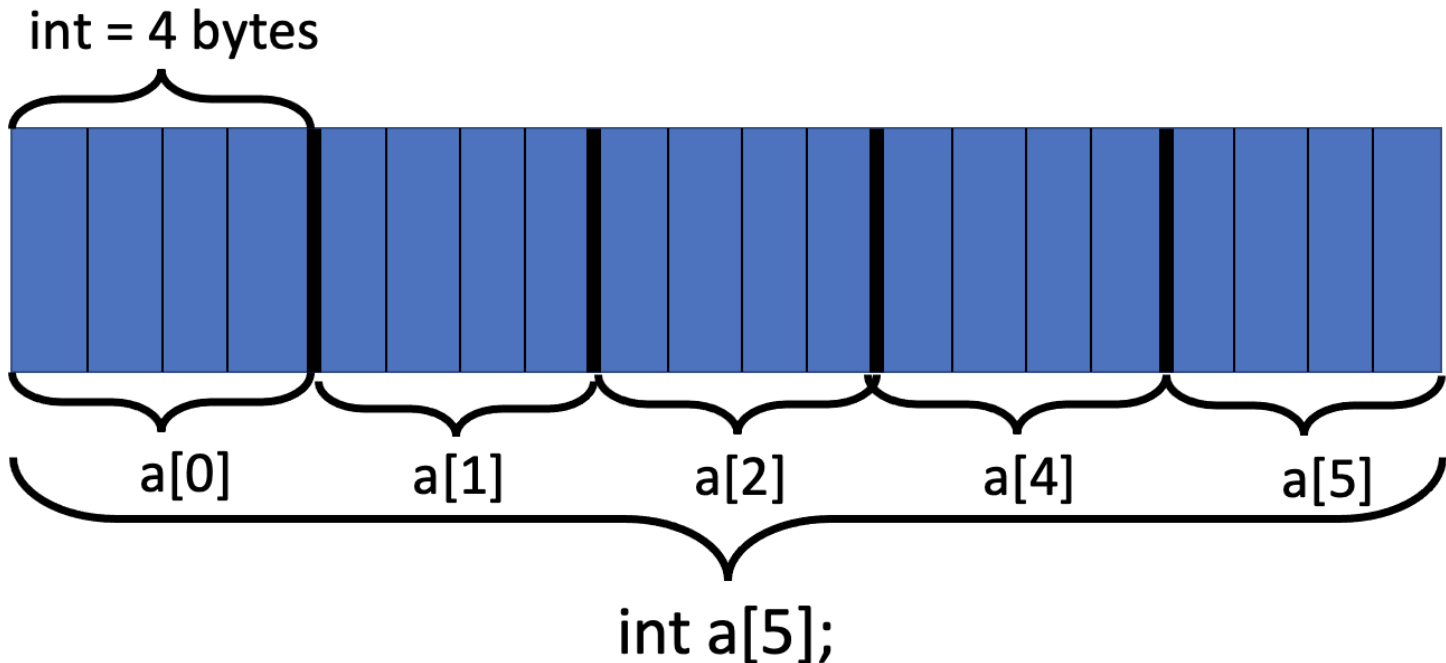


# Array

- Array declaration only reserves memory space for the array. The array will not be initialized automatically.
- If you do not initialize an array yourself, **it will contain rubbish value**, similar how variable declaration is handled in C.
- You can initialize an array using the syntax:
  - `data_type variable_name[] = {elements}; .`
  - `int a[] = {1,2,3};`
  - No need to specify the `array_size` .

# Array's Memory Layout

- Array is stored in a **contiguous** section memory.



# Vector Addition, Revisited

```
#include <stdio.h>
void main(){
    //declare and initialize array a and b.
    double a[] = {1.0, 2.0, 3.0},
           b[] = {2.0, 3.0, 4.0};
    double c[3];
    //addition
    for(int i = 0; i < 3; i=i+1){
        c[i] = a[i] + b[i];
    }
    //display each element in the array c
    for(int i = 0; i < 3; i=i+1){
        printf("%f\n", c[i]);
    }
}
```

# Increment

- Since we are going to write `i = i + 1` a lot,
- Use `i++` as a shorthand for `i = i + 1`.

```
for(int i = 0; i < 3; i++){  
    printf("%f\n", c[i]);  
}  
// is the same as  
for(int i = 0; i < 3; i = i + 1){  
    printf("%f\n", c[i]);  
}
```

- Similarly, `i += k` is short for `i = i + k`.

```
for(int i = 1; i < 10; i += 2){  
    printf("%d\n", i);  
}  
//prints out 1,3,5,7,9
```

# Today's Agenda

- What is Array?
  - How to
    - create an array?
    - access an array?
  - How an array is stored in the computer memory?
- **How to pass an array as an input argument of a function?**
  - Passing by value vs Passing by reference

# Array as Input Argument

- You can pass array as input variables of a function.
- Simply write `datatype variable_name[]` inside the parenthesis following the function name.
- `array_size` is not needed.

# Dot Product

- Consider a function `dot` computes the dot product between two vectors  $a, b$ :  $a \cdot b = \sum_i a_i b_i$ .

```
//compute dot product between a and b.  
//a and b are two input arrays  
double dot(double a[], double b[]){  
    double s = 0;  
    for(int i = 0; i < 3; i++){  
        s += a[i]*b[i];  
    }  
    return s;  
}
```

- What if you do not know the size of `a` and `b`?

# Dot Product

- Pass another input argument, specifying the array length.

```
//compute dot product between a and b.  
//a and b are two input arrays  
//len is the length of both a and b.  
double dot(double a[], double b[], int len){  
    double s = 0;  
    for(int i = 0; i < len; i++){  
        s += a[i]*b[i];  
    }  
    return s;  
}
```



# Pass by Value

When you pass an input argument to a function, you are passing by value: The program will copy the value to the input variable.

```
#include <stdio.h>
double square(double a){
    a = a*a;
    return a;
}
void main(){
    double n = 2;
    double nn = square(n);
    printf("%f %f\n", nn, n);
    //display 4 2
}
```

The value of `n` is copied to the input argument `a`, thus operations on `a` has no effect on `n`.

# Pass by Reference

- However, comparing to ordinary variables, the array occupies a much bigger memory space, thus pass by value can be expensive.
- In C, array is passed by reference.
  - If callee changes the array, caller's array will also be changed.

# Pass by Reference, Example

```
//add all elements in an array by 1
void addone(double a[], int len){
    for(int i = 0; i< len; i++){
        a[i] += 1;
    }
}
void main(){
    double a[] = {1.0, 2.0};
    addone(a,2);
    printf("%f %f\n", a[0], a[1]);
    //display 2 3, NOT 1, 2!!
}
```

# Return an Array

- Array cannot be returned by a function.
- However, since a function can make changes to caller's array, you can pass an array as input argument, and store results in that array.

```
//compute a+b and store the result in c
void add(double a[], double b[], double c[], int len){
    for(int i = 0; i < len; i++){
        c[i] = a[i] + b[i];
    }
}

void main(){
    double a[] = {1.0, 2.0}, b[] = {2.0, 3.0};
    double c[2];
    add(a,b,c,2);
    printf("%f %f\n", c[0], c[1]);
    //display 3 5
}
```

# String = `char` Array!

In C, a string is simply a `char` array that ends with `0`.

For example,

```
char a[] = {'A', 'B', 'C', 0}; // Notice the 0 at the end
char b[] = "ABC";
printf("%s\n", a);
printf("%s\n", b);
printf("%s\n", "ABC");
//all prints out "ABC"
```

# Homework

1. Download the labpack and read code introduced in the lecture,

- `vecadd.c` : Vector Addition
  - `vecdot.c` : Vector Dot Product
  - `passingbyval.c` : Passing by Value Example
  - `passingbyref.c` : Passing by Reference Example
- Make sure you understand how to
    - Create array
    - Initialize array
    - Read/Write elements of an array
    - Pass array as input to a function and
    - What is pass by value/reference.

# Lab 1, Find Max (submit)

1. Create a new C file (file->new file... save as `ab1234.c` )
2. In the `main` function, create an integer array and initialize it with a sequence of integers `[2,1,3,4,3]` .
  - Use the initialization syntax introduced in the lecture.
3. Write a function called `max` taking two inputs:
  - The input array
  - The length of the input array.
4. `max` returns the maximum value in the input array.
5. For example, provided with an array `[1,2,2]` , `max` should return 2.
6. Test `max` function in `main` using the array you just created at the 2nd step and print out the maximum value.

- Hint: write pseudo code on a paper first and talk to your coursemates/TA if you are not sure where to start.



# Lab 2, Swap (submit)

1. Write a new function in the same file called `swap`.
2. `swap` takes an array with length 2 as input.
3. `swap` does not return anything.
4. After `swap` function is called, the elements in the input array will be swapped.
  - If the input array is `[1,2]`, it becomes `[2,1]` after `swap` is called.
5. Test your `swap` function in `main` with the array `[1987,10]` and print out the array after swap.

# Lab 3, Row Major Matrix (submit)

1. Create an integer array and initialize it with value `[1,2,3,4,5,6]` in your `main` function.
2. Suppose this is a 3 by 2 matrix stored in a [row major order](#). Read the link and see what row major means.
3. Write a function `print` taking 3 inputs.
  - an array storing a matrix in row major order.
  - the number of rows of the matrix.
  - the number of columns of the matrix.
4. `print` prints out the input matrix in a proper format.
5. Test your `print` function in `main` with the array you created in the first step.

- Submit your c file with the naming convention you used in CW1.