RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Computer Architecture (CS-211)
Recitation 4

Siwei Mai

Topics

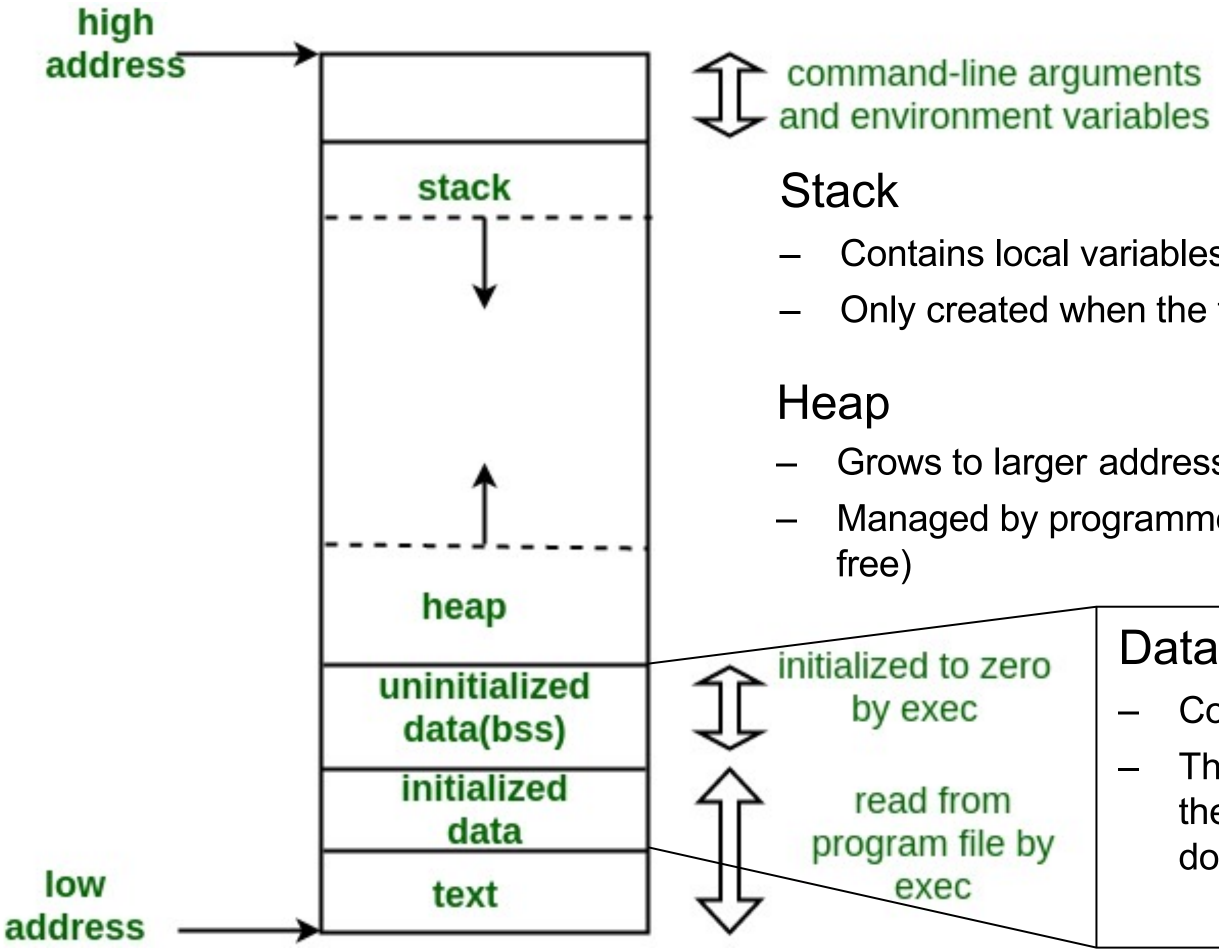
- Memory Structure in C
- Dynamic Memory Allocation
- Double Pointers
- Structure and Typedef

* Some materials are collected and compiled from previous year's CS 211 lectures and TAs

Memory Structure in C

- Stack, Heap, and Data

Reference link: https://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap


high address
command-line arguments and environment variables
stack
heap
uninitialized data(bss)
initialized to zero by exec
initialized data
read from program file by exec
text
low address

Stack

- Contains local variables and parameters
- Only created when the function is called

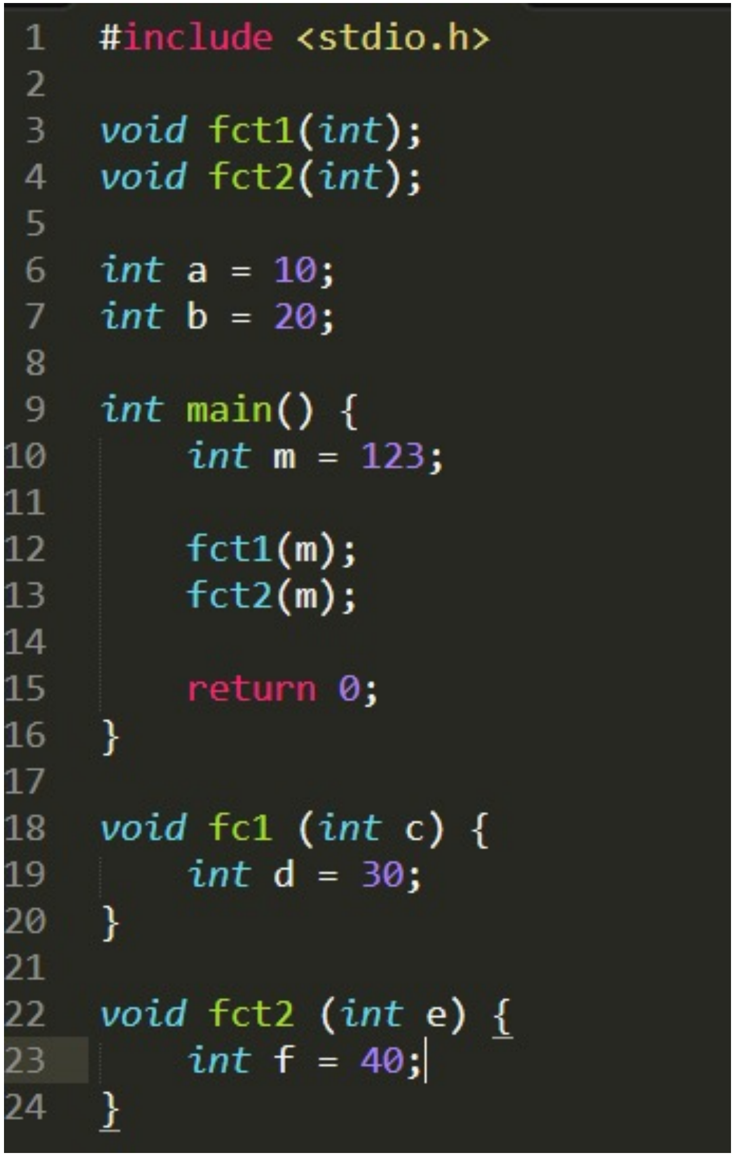
Heap

- Grows to larger addresses
- Managed by programmer (malloc, realloc, and free)

Data segment

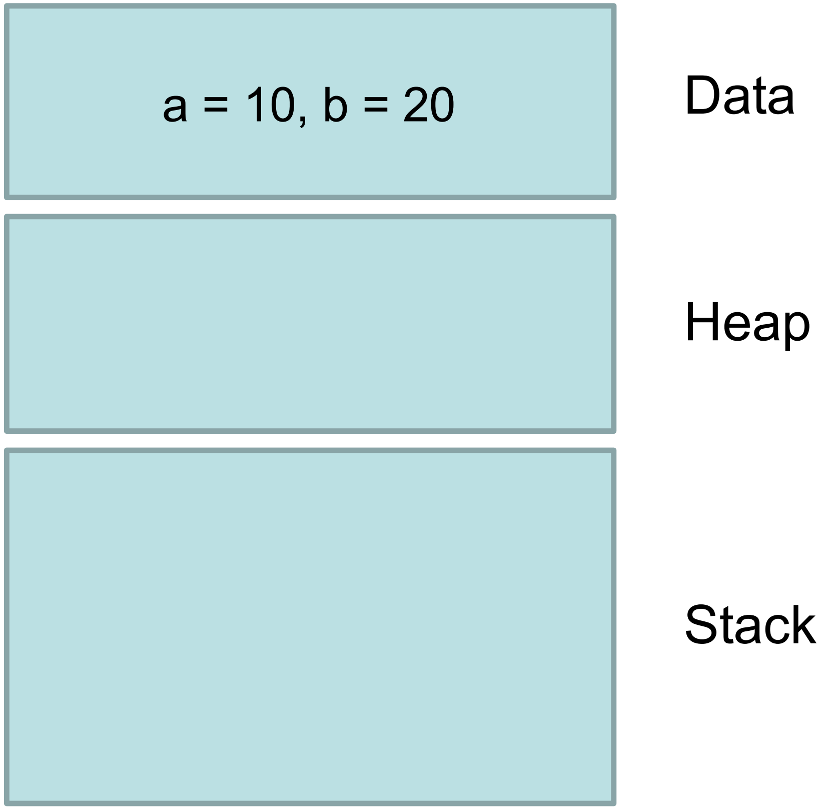
- Contains any global or static variables
- The size is determined by the size of the values in the program code, and does not change at run time

Memory flow example



```
1   #include <stdio.h>
2
3   void fct1(int);
4   void fct2(int);
5
6   int a = 10;
7   int b = 20;
8
9   int main() {
10      int m = 123;
11
12      fct1(m);
13      fct2(m);
14
15      return 0;
16  }
17
18  void fc1 (int c) {
19      int d = 30;
20  }
21
22  void fct2 (int e) {
23      int f = 40;
24  }
```

Line 6,7 : global variable



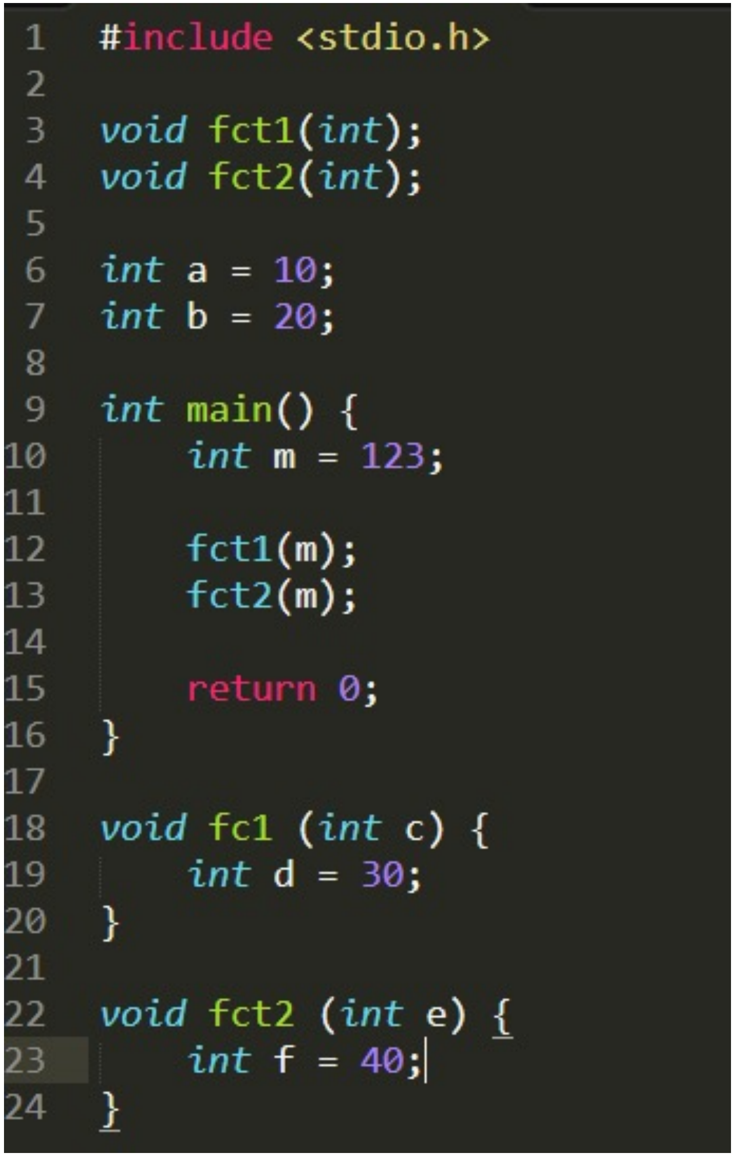
a = 10, b = 20

Data

Heap

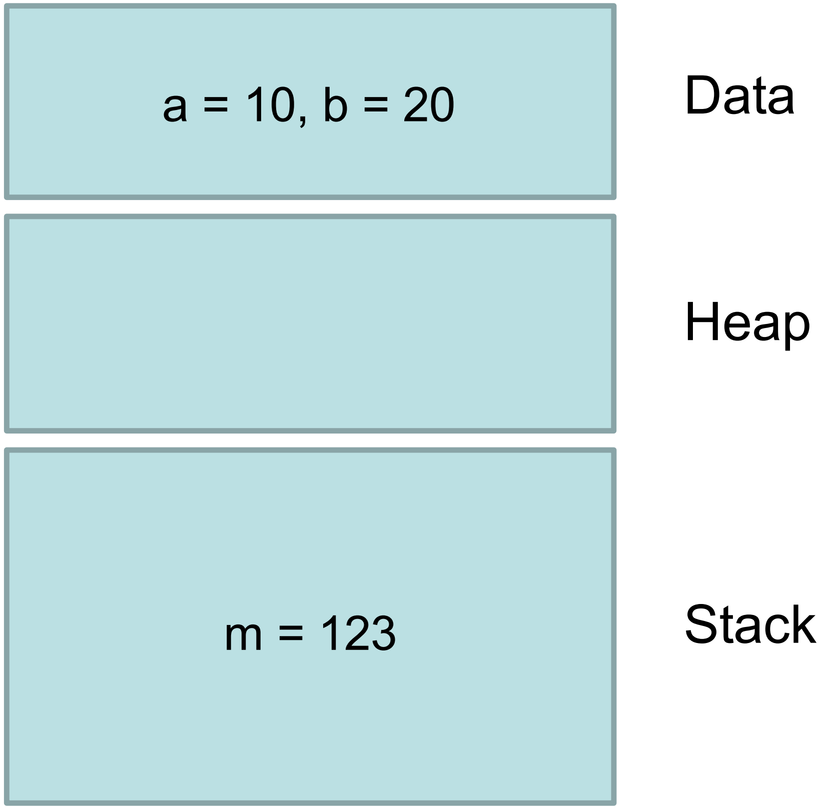
Stack

Memory flow example

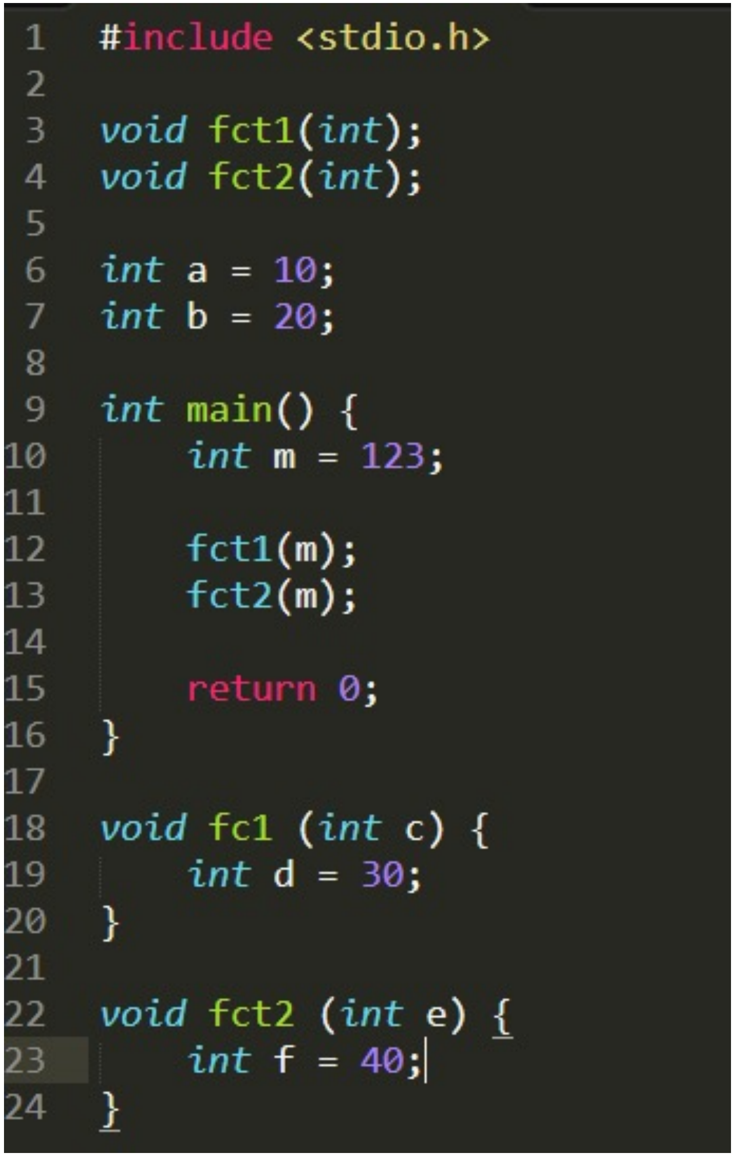


```
1   #include <stdio.h>
2
3   void fct1(int);
4   void fct2(int);
5
6   int a = 10;
7   int b = 20;
8
9   int main() {
10          int m = 123;
11
12          fct1(m);
13          fct2(m);
14
15          return 0;
16  }
17
18  void fc1 (int c) {
19          int d = 30;
20  }
21
22  void fct2 (int e) {
23          int f = 40;
24  }
```

Line 10 : local variable

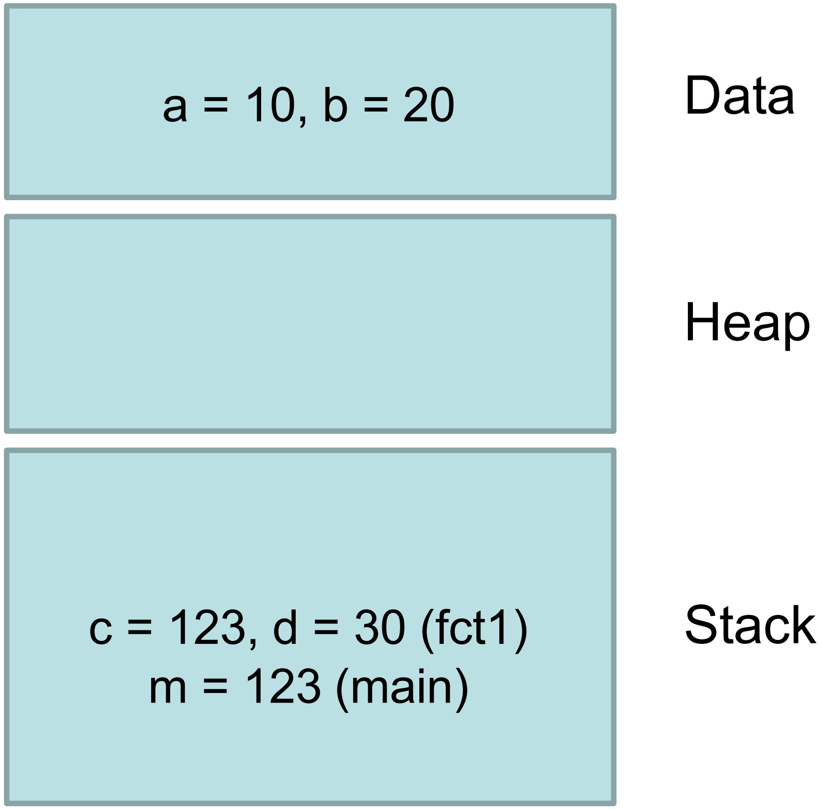


Memory flow example



```
1   #include <stdio.h>
2
3   void fct1(int);
4   void fct2(int);
5
6   int a = 10;
7   int b = 20;
8
9   int main() {
10      int m = 123;
11
12      fct1(m);
13      fct2(m);
14
15      return 0;
16  }
17
18  void fc1 (int c) {
19      int d = 30;
20  }
21
22  void fct2 (int e) {
23      int f = 40;
24  }
```

Line 10, 12 : local variable



a = 10, b = 20

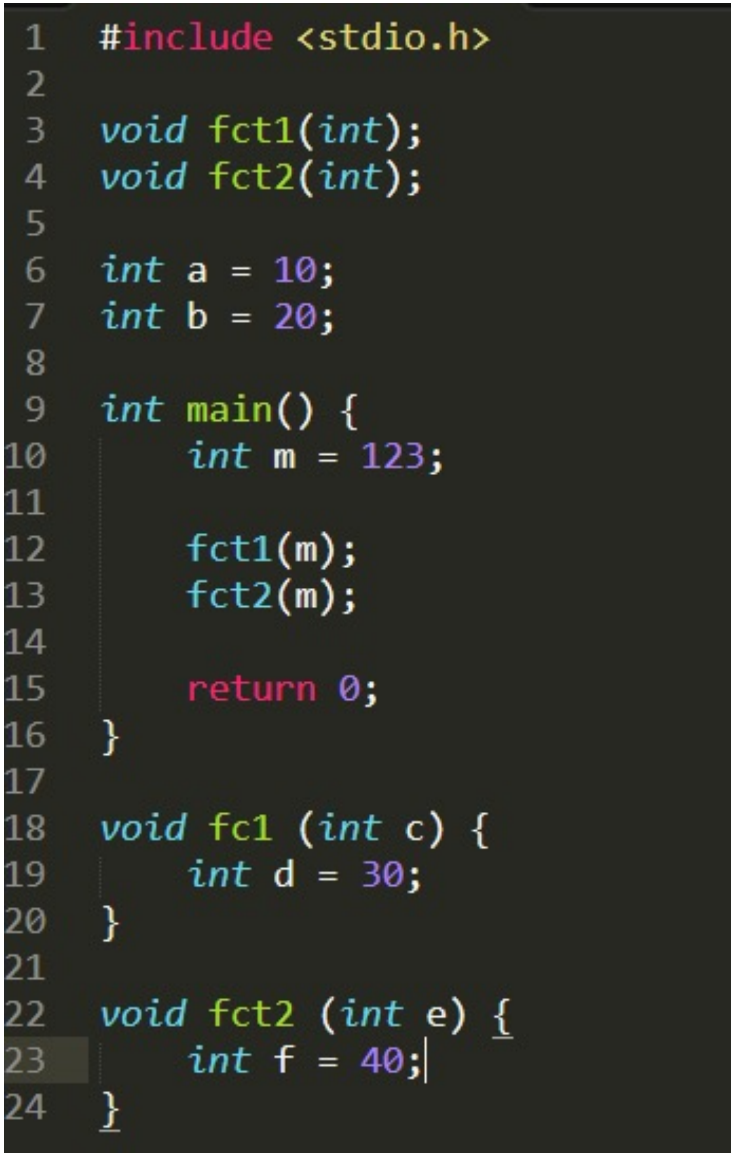
Data

Heap

c = 123, d = 30 (fct1)
m = 123 (main)

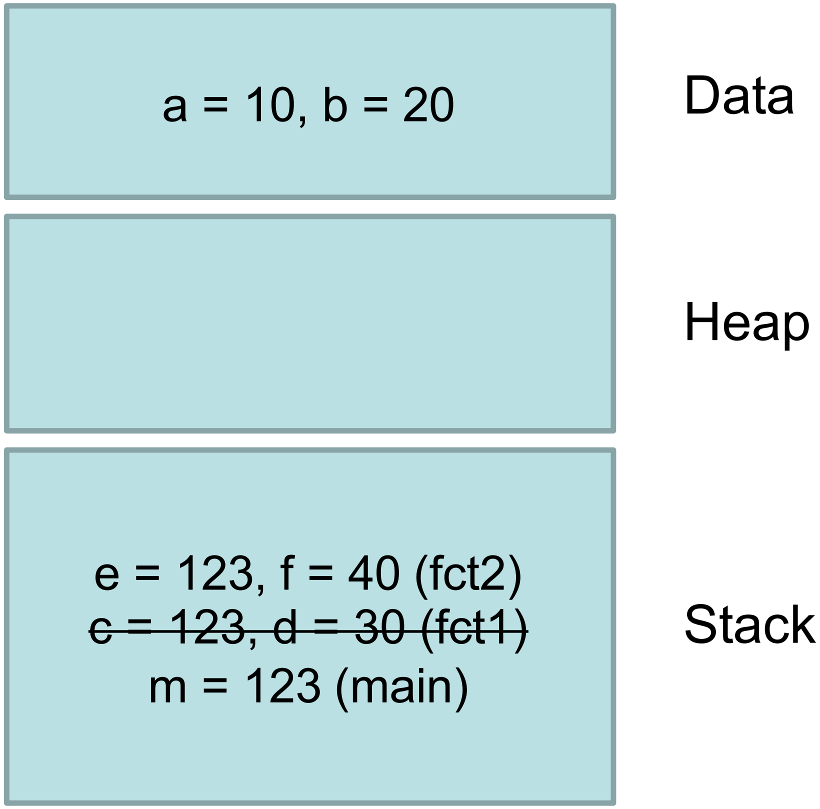
Stack

Memory flow example



```
1   #include <stdio.h>
2
3   void fct1(int);
4   void fct2(int);
5
6   int a = 10;
7   int b = 20;
8
9   int main() {
10      int m = 123;
11
12      fct1(m);
13      fct2(m);
14
15      return 0;
16  }
17
18  void fc1 (int c) {
19      int d = 30;
20  }
21
22  void fct2 (int e) {
23      int f = 40;
24  }
```

Line 10, 13 : local variable



a = 10, b = 20

Data

Heap

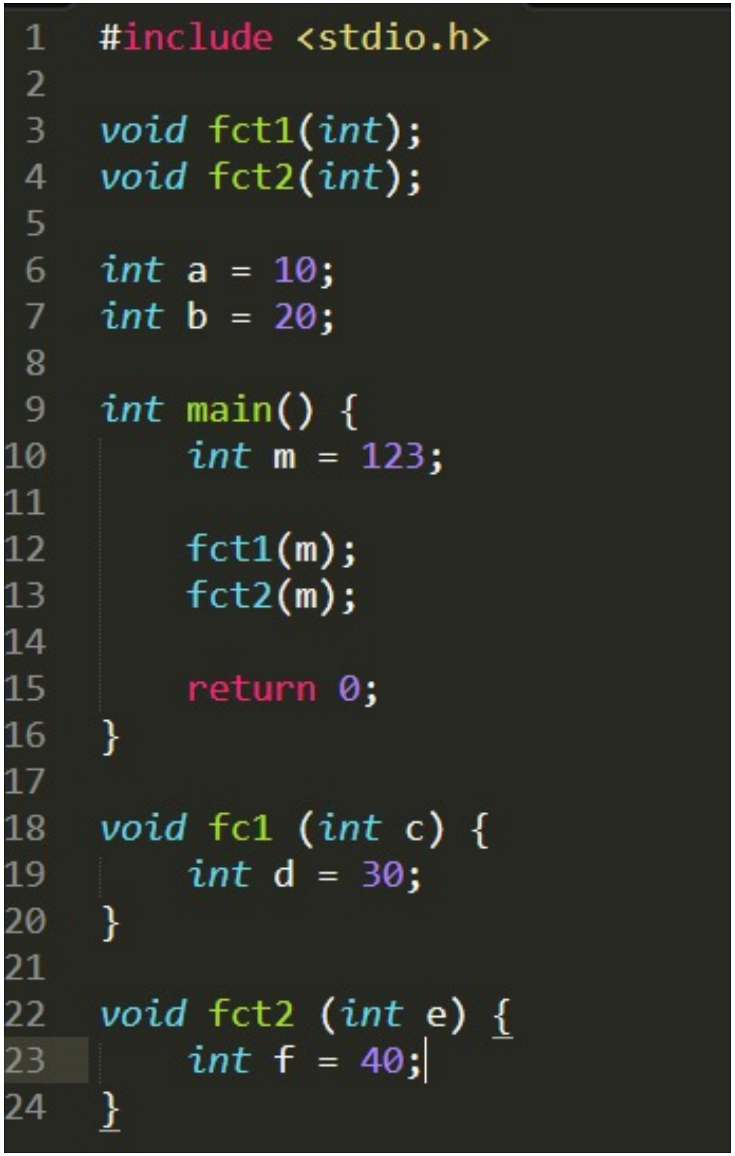
e = 123, f = 40 (fct2)

~~c = 123, d = 30 (fct1)~~

m = 123 (main)

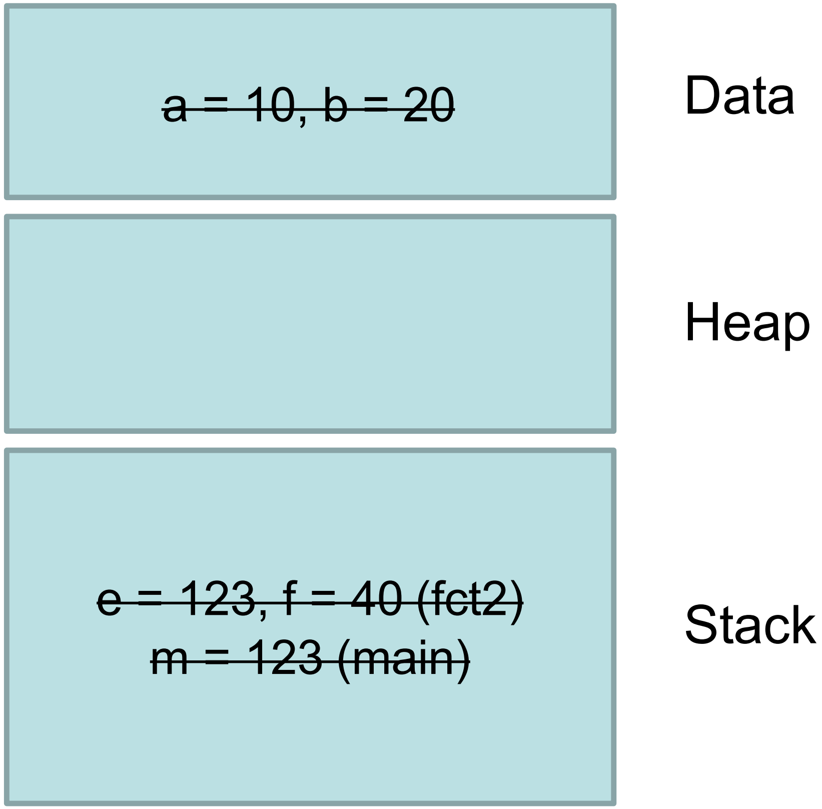
Stack

Memory flow example



```c
#include <stdio.h>

void fct1(int);
void fct2(int);

int a = 10;
int b = 20;

int main() {
    int m = 123;

    fct1(m);
    fct2(m);

    return 0;
}

void fc1 (int c) {
    int d = 30;
}

void fct2 (int e) {
    int f = 40;
}
```

End function main, fct1, fct2



~~a = 10, b = 20~~

Data

Heap

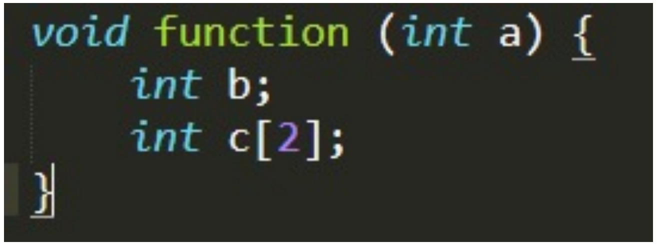
~~e = 123, f = 40 (fct2)~~
~~m = 123 (main)~~

Stack

Memory allocation

- The memory allocated in stack and data segment will be determined when it is compiling

- How much memory do we need?



```
void function (int a) {
    int b;
    int c[2];
}
```

- a-4, b-4, c-8 -> total-16 bytes

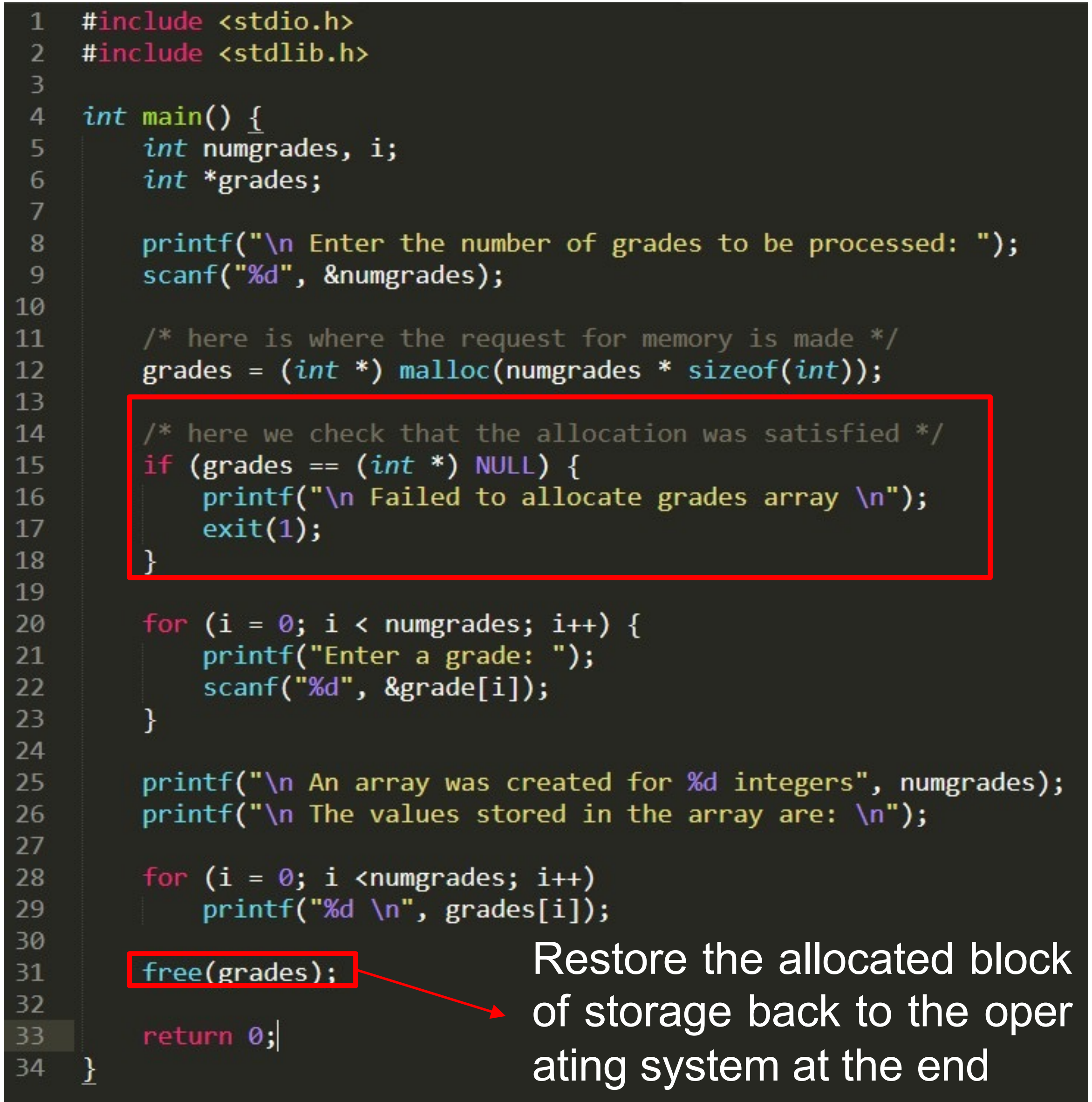
Dynamic Memory Allocation

- Allocating memory inside the **heap**
- Malloc(), realloc(), free()
- These functions are containted in <stdlib.h> header file
- To provide access to locations, malloc() returns the address of the first location that is reserved
- Address must be assigned to a pointer
- Especially useful for creating arrays

Dynamic Memory Allocation

- Malloc ()
  - Reserves the number of bytes requested by the argument passed to function
  - Returns the address of the first reserved location
  - **NULL** if there is insufficient memory
- Realloc ()
  - Changes the size of previously allocated memory to new size
- Free ()
  - Releases a block of bytes previously reserved
  - The address of the first reserved location is passed as an argument to the function

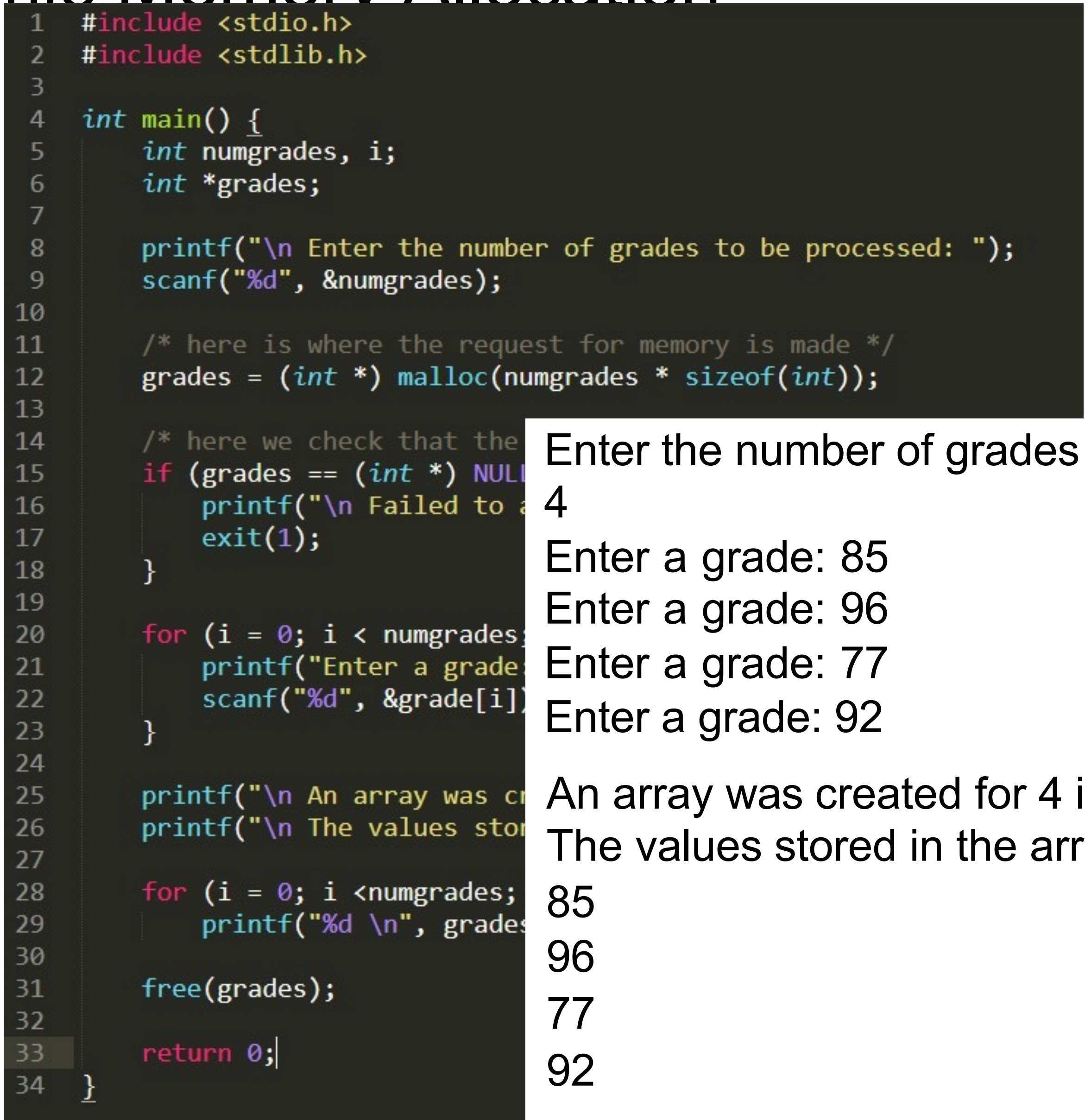
Dynamic Memory Allocation



```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int numgrades, i;
    int *grades;

    printf("\n Enter the number of grades to be processed: ");
    scanf("%d", &numgrades);

    /* here is where the request for memory is made */
    grades = (int *) malloc(numgrades * sizeof(int));

    /* here we check that the allocation was satisfied */
    if (grades == (int *) NULL) {
        printf("\n Failed to allocate grades array \n");
        exit(1);
    }

    for (i = 0; i < numgrades; i++) {
        printf("Enter a grade: ");
        scanf("%d", &grade[i]);
    }

    printf("\n An array was created for %d integers", numgrades);
    printf("\n The values stored in the array are: \n");

    for (i = 0; i <numgrades; i++)
        printf("%d \n", grades[i]);

    free(grades);

    return 0;
}
```

Restore the allocated block of storage back to the oper ating system at the end

Dynamic Memory Allocation



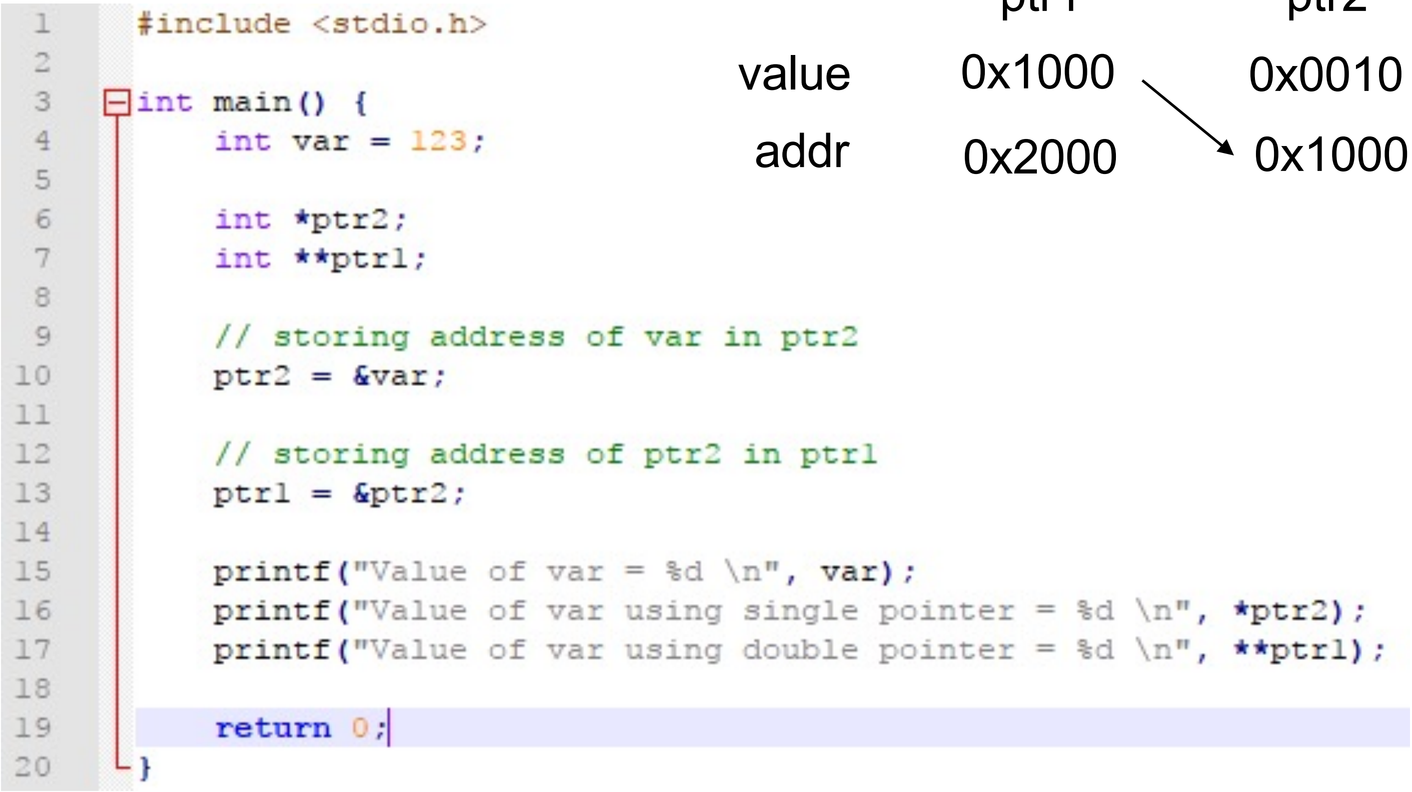
```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int numgrades, i;
    int *grades;

    printf("\n Enter the number of grades to be processed: ");
    scanf("%d", &numgrades);

    /* here is where the request for memory is made */
    grades = (int *) malloc(numgrades * sizeof(int));

    /* here we check that the
    if (grades == (int *) NUL
        printf("\n Failed to a
        exit(1);
    }

    for (i = 0; i < numgrades;
        printf("Enter a grade
        scanf("%d", &grade[i])
    }

    printf("\n An array was cr
    printf("\n The values stor

    for (i = 0; i <numgrades;
        printf("%d \n", grades

    free(grades);

    return 0;
}
```

```
Enter the number of grades to be processed:
4
Enter a grade: 85
Enter a grade: 96
Enter a grade: 77
Enter a grade: 92

An array was created for 4 integers
The values stored in the array are:
85
96
77
92
```

Double Pointer

- Declaring a pointer to pointer
  - int **ptr;

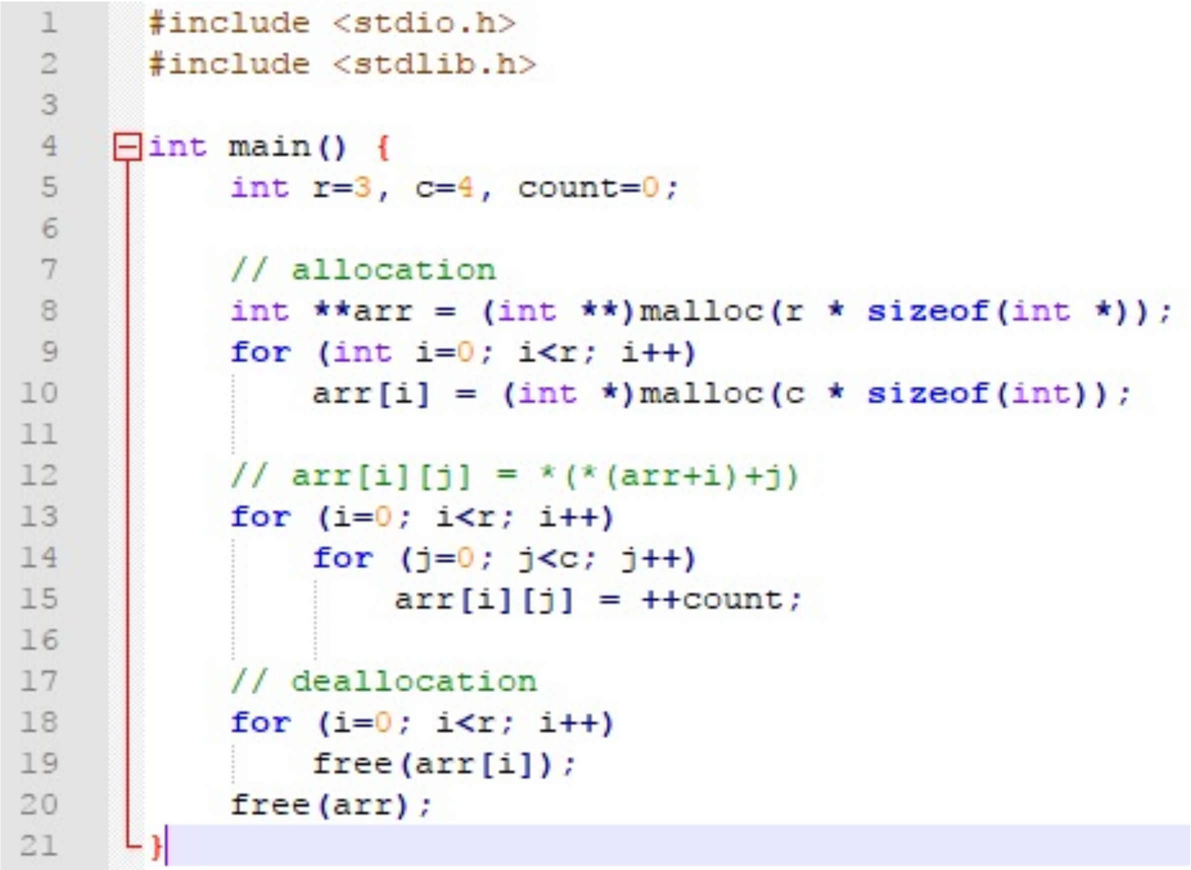
	ptr1	ptr2	var
value	0x1000	0x0010	123
addr	0x2000	0x1000	0x0010



```
#include <stdio.h>

int main() {
    int var = 123;

    int *ptr2;
    int **ptr1;

    // storing address of var in ptr2
    ptr2 = &var;

    // storing address of ptr2 in ptr1
    ptr1 = &ptr2;

    printf("Value of var = %d \n", var);
    printf("Value of var using single pointer = %d \n", *ptr2);
    printf("Value of var using double pointer = %d \n", **ptr1);

    return 0;
}
```

2D Memory Allocation

- Using double pointer



```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int r=3, c=4, count=0;

    // allocation
    int **arr = (int **)malloc(r * sizeof(int *));
    for (int i=0; i<r; i++)
        arr[i] = (int *)malloc(c * sizeof(int));

    // arr[i][j] = *(*(arr+i)+j)
    for (i=0; i<r; i++)
        for (j=0; j<c; j++)
            arr[i][j] = ++count;

    // deallocation
    for (i=0; i<r; i++)
        free(arr[i]);
    free(arr);
}
```

Structure

- Complex data type declaration that defines a physically grouped list of variables under one name
- Structure definition in C

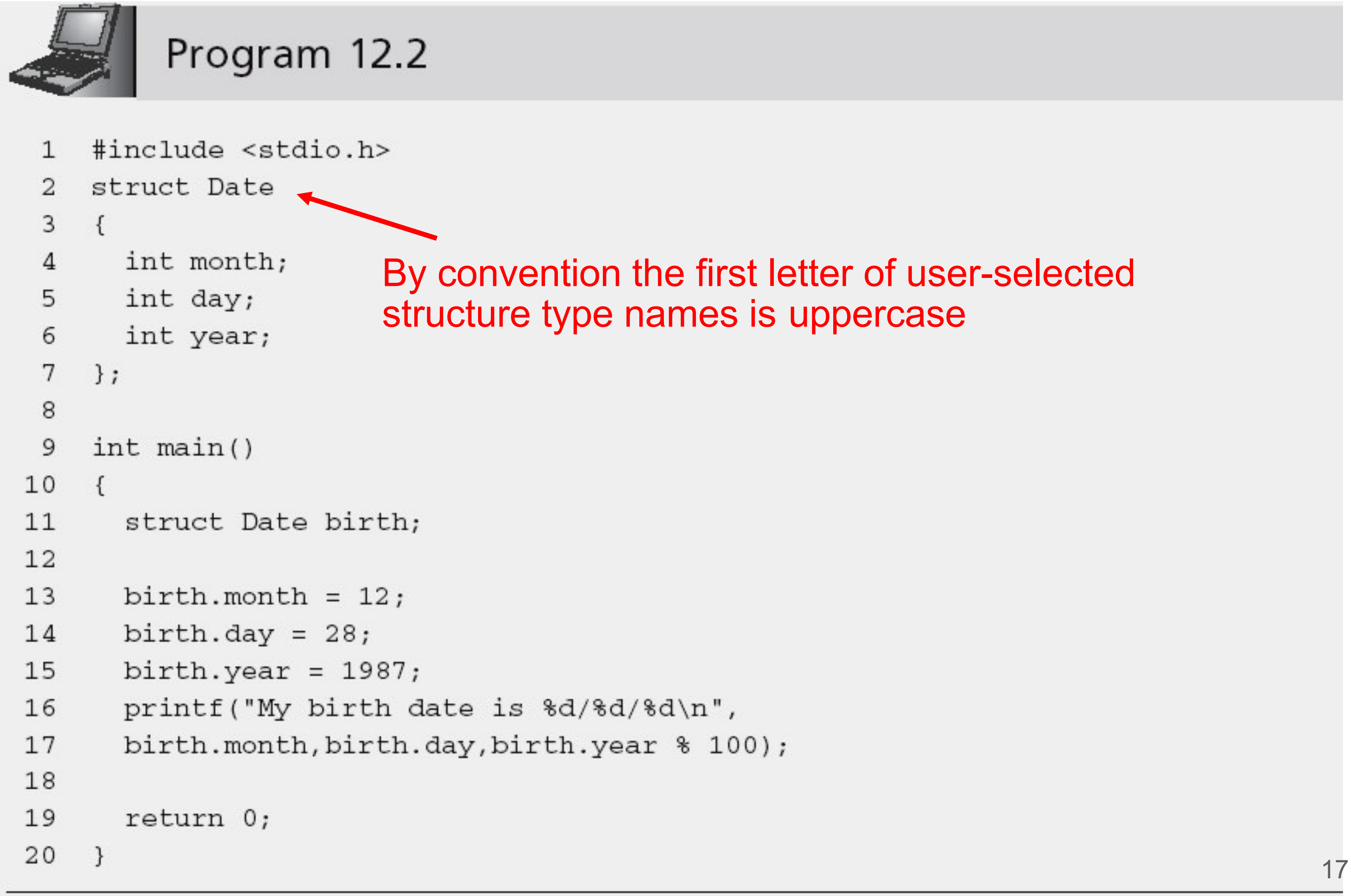
```
struct {
        int month;
        int day;
        int year;
} birth;
```

```
struct Date {
     int month;
     int day;
     int year;
};
struct Date birth, current;
```

```
struct {int month; int day; int year; } birth, current,  …;
```

  – Reserves storage for the individual data items listed in the structure
  – Three data items are the members of the structure

Structure



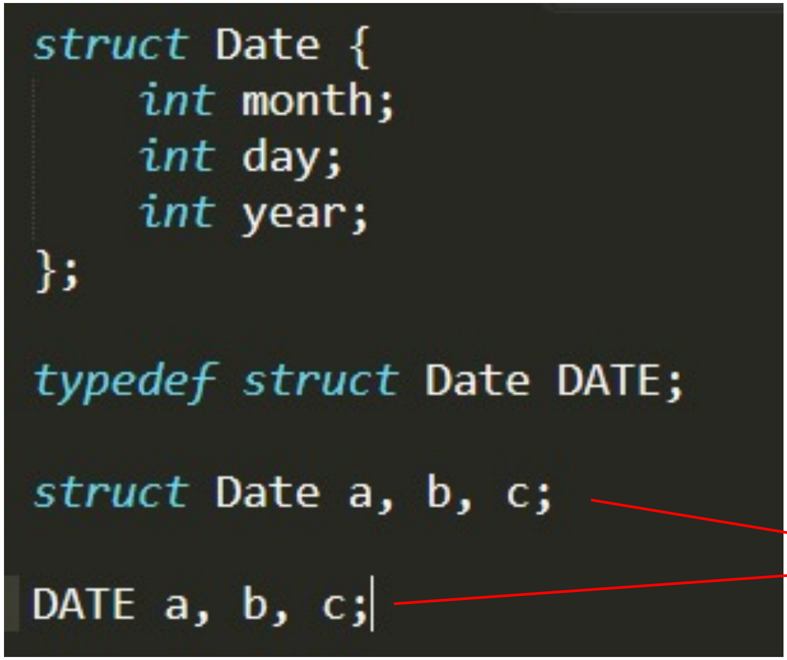
Program 12.2

```
#include <stdio.h>
struct Date
{
  int month;
  int day;
  int year;
};

int main()
{
  struct Date birth;

  birth.month = 12;
  birth.day = 28;
  birth.year = 1987;
  printf("My birth date is %d/%d/%d\n",
  birth.month,birth.day,birth.year % 100);

  return 0;
}
```

By convention the first letter of user-selected structure type names is uppercase

Typedef Statement

- A commonly used programming technique when dealing with structure declarations



```
struct Date {
    int month;
    int day;
    int year;
};

typedef struct Date DATE;

struct Date a, b, c;

DATE a, b, c;
```

These two statements are same

Q&A

Thanks!