CSCB09 Software Tools and Systems Programming
Memory Model / IO

Marcelo Ponce

Winter 2025

Department of Computer and Mathematical Sciences - UTSC

Today's lecture

Memory Model (cont.)

- Memory Issues
- Memory Model worksheet

IO

- Quick recap
- IO Worksheet

Memory Model (cont.)

The *Complete* Memory Model

Static data

Space for the *evil* **global** variables and variables declared as **static**

Dynamic data (Heap)

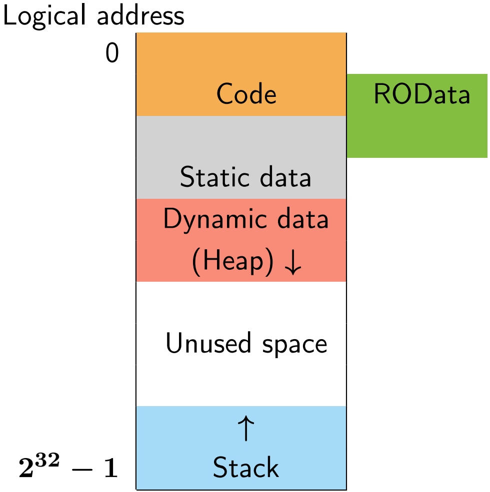
Space for dynamically allocated data structures (`malloc`, `calloc`).

Stack

Space for variables created in function calls: a function's parameters and a function's local variables

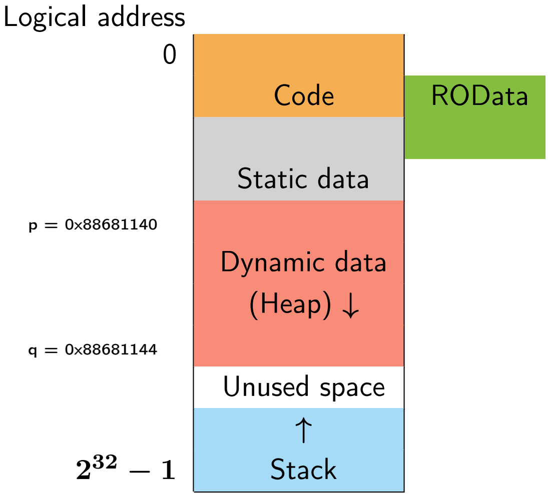
Code/Text

string literals: ROData (read-only data) code/text, or static data depending on platform



- Suppose that,

```
p = malloc(...);
q = malloc(...);
```

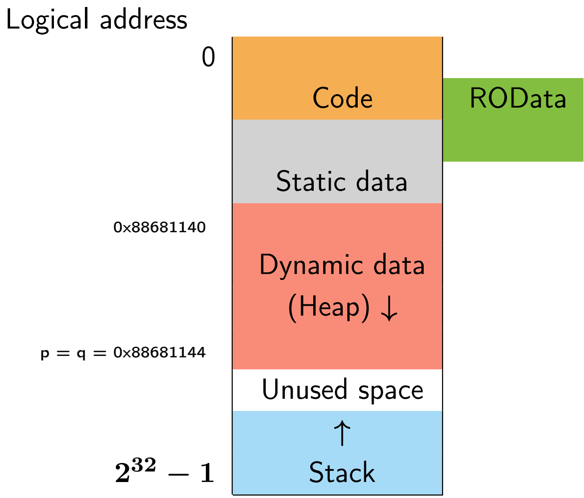


- Suppose that,

```
p = malloc(...);
q = malloc(...);
```

- Next consider,

```
p = q;
```



Memory Leaks

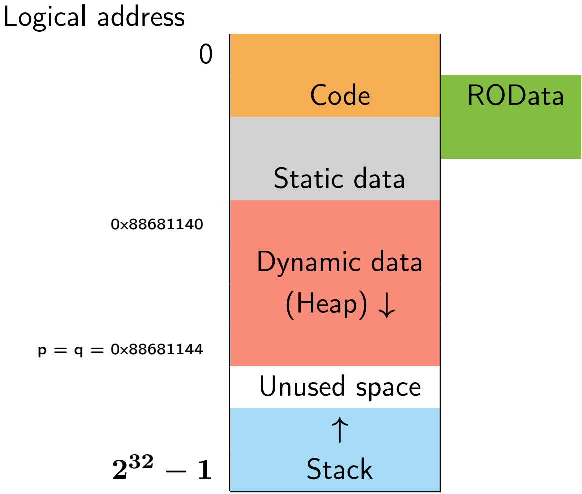
- Suppose that,

```
p = malloc(...);
q = malloc(...);
```

- Next consider,

```
p = q;
```

- What happens?



Memory Leaks

- Suppose that,

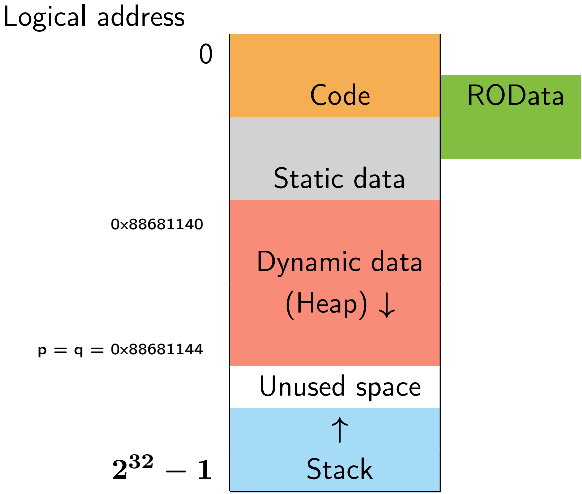
  `p = malloc(...);`

  `q = malloc(...);`

- Next consider,

  `p = q;`

- **Problem**: we have no way of accessing `p`'s old block.
- We also have no way of freeing `p`'s old block.



Memory Leaks

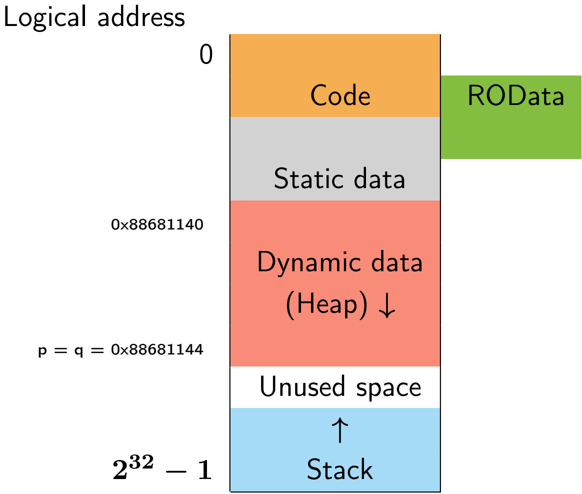
- Suppose that,

```
p = malloc(...);
q = malloc(...);
```

- Next consider,

```
p = q;
```

- **Problem**: we have no way of accessing `p`'s old block.
- We also have no way of freeing `p`'s old block.
- This is called a **memory leak**. One of the most common programming errors.



Dangling Pointers

```c
char *p = malloc(5);
...
free(p);
```

p is now a pointer to memory it does not own.

```c
strcpy(p,"abcd");    // Bad things can
    happen
```

Memory Model worksheet

Code Fragment	Space?	Where?	De-allocated when?

<pre>int main() {
  int i;
}</pre>			

<pre>int fun() {
  float i;
}
 	
int main() {
  fun();
}</pre>			

<pre>int fun(char i)
     {
 	
  . . .
}
 	
int main() {
  fun('a');
}</pre>			

Memory Model worksheet

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	`sizeof(int)`		

`int fun() {`
`  float i;`
`}`
 	
`int main() {`
`  fun();`
`}`			

`int fun(char i)`
`    {`
 	
`  . . .`
`}`
 	
`int main() {`
`  fun('a');`
`}`			

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	`sizeof(int)`	Stack	

`int fun() {`
`  float i;`
`}`
 	
`int main() {`
`  fun();`
`}`			

`int fun(char i)`
`    {`
 	
`  . . .`
`}`
 	
`int main() {`
`  fun('a');`
`}`			

Memory Model worksheet

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  int i;
}</pre></td><td><code>sizeof(int)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int fun() {
  float i;
}

int main() {
  fun();
}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int fun(char i)
    {

  . . .
}

int main() {
  fun('a');
}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  int i;
}</pre></td><td><code>sizeof(int)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int fun() {
  float i;
}

int main() {
  fun();
}</pre></td><td><code>sizeof(float)</code></td><td></td><td></td></tr>
<tr><td><pre>int fun(char i)
    {

  . . .
}

int main() {
  fun('a');
}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  int i;
}</pre></td><td><code>sizeof(int)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int fun() {
  float i;
}

int main() {
  fun();
}</pre></td><td><code>sizeof(float)</code></td><td>Stack</td><td></td></tr>
<tr><td><pre>int fun(char i)
    {

  . . .
}

int main() {
  fun('a');
}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	`sizeof(int)`	Stack	when program ends

`int fun() {`
`  float i;`
`}`
 	
`int main() {`
`  fun();`
`}`
	`sizeof(float)`	Stack	when `fun` returns

`int fun(char i)`
`    {`
 	
`  . . .`
`}`
 	
`int main() {`
`  fun('a');`
`}`			

Memory Model worksheet

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	`sizeof(int)`	Stack	when program ends

`int fun() {`
`  float i;`
`}`
 	
`int main() {`
`  fun();`
`}`
	`sizeof(float)`	Stack	when `fun` returns

`int fun(char i)`
`    {`
 	
`  . . .`
`}`
 	
`int main() {`
`  fun('a');`
`}`
	`1 == sizeof(char)`		

Memory Model worksheet

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  int i;
}</pre></td><td><code>sizeof(int)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int fun() {
  float i;
}

int main() {
  fun();
}</pre></td><td><code>sizeof(float)</code></td><td>Stack</td><td>when <code>fun</code> returns</td></tr>
<tr><td><pre>int fun(char i)
    {

  . . .
}

int main() {
  fun('a');
}</pre></td><td><code>1 == sizeof(char)</code></td><td>Stack</td><td></td></tr>
</table>

Memory Model worksheet

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  int i;`
`}`
	`sizeof(int)`	Stack	when program ends

`int fun() {`
`  float i;`
`}`
 	
`int main() {`
`  fun();`
`}`
	`sizeof(float)`	Stack	when `fun` returns

`int fun(char i)`
`    {`
 	
`  . . .`
`}`
 	
`int main() {`
`  fun('a');`
`}`
	`1 == sizeof(char)`	Stack	when `fun` returns

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
   char i[10] =
         "hello";

}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
   char *i;

}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
   int *i;

}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
   char *i = "
         hello";

}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  char i[10] = "hello";`
`}`			

`int main() {`
`  char *i;`
`}`
	`sizeof(char *)`		

`int main() {`
`  int *i;`
`}`			

`int main() {`
`  char *i = "hello";`
`}`			

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre><code>int main() {
  char i[10] =
        "hello";

}</code></pre></td><td></td><td></td><td></td></tr>
<tr><td><pre><code>int main() {
  char *i;

}</code></pre></td><td><code>sizeof(char *)</code></td><td>Stack</td><td></td></tr>
<tr><td><pre><code>int main() {
  int *i;

}</code></pre></td><td></td><td></td><td></td></tr>
<tr><td><pre><code>int main() {
  char *i = "
        hello";

}</code></pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  char i[10] =
        "hello";

}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
  char *i;

}</pre></td><td><code>sizeof(char *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;

}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
  char *i = "
        hello";

}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  char i[10] =`
`      "hello";`
 	
`}`			

`int main() {`
`  char *i;`
 	
`}`
	`sizeof(char *)`	Stack	when program ends

`int main() {`
`  int *i;`
 	
`}`
	`sizeof(int *)`		

`int main() {`
`  char *i = "`
`      hello";`
 	
`}`			

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  char i[10] =
      "hello";

}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
  char *i;

}</pre></td><td><code>sizeof(char *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;

}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td></td></tr>
<tr><td><pre>int main() {
  char *i = "
      hello";

}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  char i[10] =
      "hello";
}</pre></td><td></td><td></td><td></td></tr>
<tr><td><pre>int main() {
  char *i;
}</pre></td><td><code>sizeof(char *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  char *i = "
      hello";
}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  char i[10] =
      "hello";
}</pre></td><td><code>10*sizeof(char) == 10</code></td><td></td><td></td></tr>
<tr><td><pre>int main() {
  char *i;
}</pre></td><td><code>sizeof(char *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  char *i = "
      hello";
}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  char i[10] =`
`      "hello";`
 	
`}`
	`10*sizeof(char) ==`
	
	`10`
		Stack	

`int main() {`
`  char *i;`
 	
`}`
	`sizeof(char *)`	Stack	when program ends

`int main() {`
`  int *i;`
 	
`}`
	`sizeof(int *)`	Stack	when program ends

`int main() {`
`  char *i = "`
`      hello";`
 	
`}`			

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  char i[10] =`
`      "hello";`
`}`
	`10*sizeof(char) ==`
	`10`
		Stack	when program ends

`int main() {`
`  char *i;`
`}`
	`sizeof(char *)`	Stack	when program ends

`int main() {`
`  int *i;`
`}`
	`sizeof(int *)`	Stack	when program ends

`int main() {`
`  char *i = "`
`      hello";`
`}`			

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int main() {
  char i[10] =
      "hello";
}</pre></td><td><code>10*sizeof(char) == 10</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  char *i;
}</pre></td><td><code>sizeof(char *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  char *i = "
      hello";
}</pre></td><td><code>sizeof(char *) ; 6 == 6*sizeof(char)</code></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?
`int main() { char i[10] = "hello"; }`	`10*sizeof(char) == 10`	Stack	when program ends
`int main() { char *i; }`	`sizeof(char *)`	Stack	when program ends
`int main() { int *i; }`	`sizeof(int *)`	Stack	when program ends
`int main() { char *i = "hello"; }`	`sizeof(char *) ; 6 == 6*sizeof(char)`	Stack; read-only memory	

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int main() {`
`  char i[10] =`
`      "hello";`
`}`
	`10*sizeof(char) == 10`	Stack	when program ends

`int main() {`
`  char *i;`
`}`
	`sizeof(char *)`	Stack	when program ends

`int main() {`
`  int *i;`
`}`
	`sizeof(int *)`	Stack	when program ends

`int main() {`
`  char *i = "`
`      hello";`
`}`
	`sizeof(char *) ; 6 == 6*sizeof(char)`	Stack; read-only memory	when program ends; when program ends

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

<pre>int fun(int *i) {
  . . .
}
 	
int main() {
  int i[5] =
      {4,5,2,5,1};
 	
  fun(i);
}</pre>			

<pre>int main() {
  int *i;
  i = malloc(
      sizeof(
      int));
  ...
}</pre>			

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int fun(int *i) {`
`  . . .`
`}`
 	
`int main() {`
`  int i[5] =`
`      {4,5,2,5,1};`
 	
`  fun(i);`
`}`
	`sizeof(int *)`
	
	`5*sizeof(int)`		

`int main() {`
`  int *i;`
`  i = malloc(`
`      sizeof(`
`      int));`
`  ...`
`}`			

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?

`int fun(int *i) {`
`  . . .`
`}`
	`sizeof(int *)`	Stack	

`int main() {`
`  int i[5] = {4,5,2,5,1};`
`  fun(i);`
`}`
	`5*sizeof(int)`	Stack	

`int main() {`
`  int *i;`
`  i = malloc(sizeof(int));`
`  ...`
`}`			

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int fun(int *i)
    {
  . . .
}</pre></td><td>sizeof(int *)</td><td>Stack</td><td>when fun returns</td></tr>
<tr><td><pre>int main() {
  int i[5] =
      {4,5,2,5,1};

  fun(i);
}</pre></td><td>5*sizeof(int)</td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
  i = malloc(
      sizeof(
      int));
  ...
}</pre></td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?
`int fun(int *i) { . . . }`	`sizeof(int *)`	Stack	when fun returns
`int main() { int i[5] = {4,5,2,5,1}; fun(i); }`	`5*sizeof(int)`	Stack	when program ends
`int main() { int *i; i = malloc( sizeof( int)); ... }`	`sizeof(int *)` / `sizeof(int)`		

Memory Model worksheet (cont.)

Code Fragment	Space?	Where?	De-allocated when?
`int fun(int *i) { . . . }`	`sizeof(int *)`	Stack	when fun returns
`int main() { int i[5] = {4,5,2,5,1}; fun(i); }`	`5*sizeof(int)`	Stack	when program ends
`int main() { int *i; i = malloc( sizeof( int)); ... }`	`sizeof(int *)` / `sizeof(int)`	Stack / Heap	

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>int fun(int *i)
    {
  . . .
}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td>when fun returns</td></tr>
<tr><td><pre>int main() {
  int i[5] =
      {4,5,2,5,1};

  fun(i);
}</pre></td><td><code>5*sizeof(int)</code></td><td>Stack</td><td>when program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
  i = malloc(
      sizeof(
      int));
  ...
}</pre></td><td><code>sizeof(int *)</code> / <code>sizeof(int)</code></td><td>Stack / Heap</td><td>when program ends / when free called or when program ends</td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td>

```c
void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}
```

</td><td></td><td></td><td></td></tr>
<tr><td>

```c
int main() {
  int *i;
  fun(&i);
  free(i);
}
```

</td><td></td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td>

```c
void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}
```

```c
int main() {
  int *i;
  fun(&i);
  free(i);
}
```

</td><td>sizeof(int **) /

7*sizeof(int)</td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td>

```c
void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}
```

```c
int main() {
  int *i;
  fun(&i);
  free(i);
}
```

</td><td>sizeof(int **) /
7*sizeof(int)</td><td>Stack / Heap</td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td>

```c
void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}
```

```c
int main() {
  int *i;
  fun(&i);
  free(i);
}
```

</td><td>sizeof(int **) /
7*sizeof(int)</td><td>Stack / Heap</td><td>when function returns / when free called or program ends</td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}</pre></td><td>sizeof(int **) /
7*sizeof(int)</td><td>Stack / Heap</td><td>when function returns / when free called or program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
  fun(&i);
  free(i);
}</pre></td><td>sizeof(int *)</td><td></td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}</pre></td><td><code>sizeof(int **)</code> / <code>7*sizeof(int)</code></td><td>Stack / Heap</td><td>when function returns / when free called or program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
  fun(&i);
  free(i);
}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td></td></tr>
</table>

Memory Model worksheet (cont.)

<table>
<tr><th>Code Fragment</th><th>Space?</th><th>Where?</th><th>De-allocated when?</th></tr>
<tr><td><pre>void fun(int **
    i) {
  *i = malloc(
      sizeof(
      int) * 7)
      ;
  ...
}</pre></td><td><code>sizeof(int **)</code> / <code>7*sizeof(int)</code></td><td>Stack / Heap</td><td>when function returns / when free called or program ends</td></tr>
<tr><td><pre>int main() {
  int *i;
  fun(&i);
  free(i);
}</pre></td><td><code>sizeof(int *)</code></td><td>Stack</td><td>when program ends</td></tr>
</table>

Worksheet Q2

Write a program that declares 3 strings:

- The first named first should be set to the value "Monday", and be stored on the *stack frame* for main.
- second should be a *string literal* with the value "Tuesday".
- third should have value "Wednesday" and be on the *heap*.
- The pointers for second and third will be in *stack frame* for main.

Worksheet Q2

Write a program that declares 3 strings:

- The first named first should be set to the value "Monday", and be stored on the *stack frame* for main.
- second should be a *string literal* with the value "Tuesday".
- third should have value "Wednesday" and be on the *heap*.
- The pointers for second and third will be in *stack frame* for main.

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

int main() {
  // store in stack
  char first[] = "Monday";
  // string literal
  char *second = "Tuesday";
  // in heap
  char *third = malloc(sizeof(char) * 10);
  strncpy(third, "Wednesday", 10);

  return 0;
}
```

Worksheet Q3

- Write statements to shorten the strings to the abbreviations for the day names.
  For example, change "Monday" to "Mon".
  Which string can not be changed in place? Why not?

Worksheet Q3

- Write statements to shorten the strings to the abbreviations for the day names.
  For example, change "Monday" to "Mon".
  Which string can not be changed in place? Why not?

```
first[3] = '\0';
// second[3] = '\0'; // This will cause undefined behaviour
third[3] = '\0';
```

Worksheet Q5

- Add to your program so that it declares an array string list of 3 pointers to char and point the elements to first, second, and third, respectively.
- So now you have an array of strings. Where is the memory allocated for this array?

Worksheet Q5

- Add to your program so that it declares an array string list of 3 pointers to char and point the elements to first, second, and third, respectively.
- So now you have an array of strings. Where is the memory allocated for this array?

```
char *array[3];

array[0] = first;
array[1] = second;
array[2] = third;
```

IO

Storing Characters

- Characters are 1 byte long integers
- The ASCII code maps characters to corresponding int.
- E.g. `char c = 'A'` is identical to `char c = 65`
- When you store an 'A' the 8 bits that are stored are: 01000001 (representing the number 65 in binary).

Character Name	Char	Code	Decimal	Binary	Hex
Null	NUL	Ctrl @	0	00000000	00
Start of Heading	SOH	Ctrl A	1	00000001	01
Start of Text	STX	Ctrl B	2	00000010	02
End of Text	ETX	Ctrl C	3	00000011	03
End of Transmit	EOT	Ctrl D	4	00000100	04
Enquiry	ENQ	Ctrl E	5	00000101	05
Acknowledge	ACK	Ctrl F	6	00000110	06
Bell	BEL	Ctrl G	7	00000111	07
Back Space	BS	Ctrl H	8	00001000	08
Horizontal Tab	TAB	Ctrl I	9	00001001	09
Line Feed	LF	Ctrl J	10	00001010	0A
Vertical Tab	VT	Ctrl K	11	00001011	0B
Form Feed	FF	Ctrl L	12	00001100	0C
Carriage Return	CR	Ctrl M	13	00001101	0D
Shift Out	SO	Ctrl N	14	00001110	0E
Shift In	SI	Ctrl O	15	00001111	0F
Data Line Escape	DLE	Ctrl P	16	00010000	10
Device Control 1	DC1	Ctrl Q	17	00010001	11
Device Control 2	DC2	Ctrl R	18	00010010	12
Device Control 3	DC3	Ctrl S	19	00010011	13
Device Control 4	DC4	Ctrl T	20	00010100	14
Negative Acknowledge	NAK	Ctrl U	21	00010101	15
Synchronous Idle	SYN	Ctrl V	22	00010110	16
End of Transmit Block	ETB	Ctrl W	23	00010111	17
Cancel	CAN	Ctrl X	24	00011000	18

SRC: `https://www.eso.org/~ndelmott/ascii.html`

Capital A	A	Shift A	65	01000001	41
Capital B	B	Shift B	66	01000010	42
Capital C	C	Shift C	67	01000011	43
Capital D	D	Shift D	68	01000100	44
Capital E	E	Shift E	69	01000101	45
Capital F	F	Shift F	70	01000110	46
Capital G	G	Shift G	71	01000111	47
Capital H	H	Shift H	72	01001000	48
Capital I	I	Shift I	73	01001001	49
Capital J	J	Shift J	74	01001010	4A
Capital K	K	Shift K	75	01001011	4B
Capital L	L	Shift L	76	01001100	4C
Capital M	M	Shift M	77	01001101	4D
Capital N	N	Shift N	78	01001110	4E
Capital O	O	Shift O	79	01001111	4F
Capital P	P	Shift P	80	01010000	50
Capital Q	Q	Shift Q	81	01010001	51
Capital R	R	Shift R	82	01010010	52
Capital S	S	Shift S	83	01010011	53
Capital T	T	Shift T	84	01010100	54
Capital U	U	Shift U	85	01010101	55
Capital V	V	Shift V	86	01010110	56
Capital W	W	Shift W	87	01010111	57
Capital X	X	Shift X	88	01011000	58
Capital Y	Y	Shift Y	89	01011001	59
Capital Z	Z	Shift Z	90	01011010	5A
Left Bracket	[	[	91	01011011	5B
Backward Slash	\	\	92	01011100	5C
Right Bracket	]	]	93	01011101	5D

Binary I/O

- `fgets` reads characters, `fprintf` writes characters.
- By contrast, `fread` and `fwrite` operate on bytes.

```
size_t  fread(void *ptr, size_t size,
              size_t nmemb, FILE *
                 stream);
```

read `nmemb` × `size` bytes into memory at `ptr`

returns number of items read

```
size_t fwrite(const void *ptr, size_t
    size,
```

write `nmemb` × `size` bytes from `ptr` to the file pointer stream

returns number of items written

Binary I/O

- `fgets` reads characters, `fprintf` writes characters.
- By contrast, `fread` and `fwrite` operate on bytes.

```
size_t  fread(void *ptr, size_t size,
                 size_t nmemb, FILE *
                    stream);
```

read `nmemb` × `size` bytes into memory at `ptr`

returns number of items read

```
/* write an integer to the file */
int num = 1999999;
n = fwrite(&num, sizeof(num), 1, fp);
```

```
size_t fwrite(const void *ptr, size_t
    size,
```

write `nmemb` × `size` bytes from `ptr` to the file pointer stream

returns number of items written

```
/* read an integer from the file */
int num;
n = fread(&num, sizeof(num), 1, fp);
```

What's written to the file? (show the bits)	**Code**	**What's in the Memory?** (show the bits)
	`char ch = 'A';`	
	`fprintf(fp, "A");`	
	`char ch = 'A';` `fprintf(fp, "%c", ch);`	
	`fprintf(fp, "5");`	
	`char ch = '5';` `fprintf(fp,"%c",ch);`	
	`int i = 5;`	

(Assuming size of an integer is 4 bytes)

What's written to the file? (show the bits)	Code	What's in the Memory? (show the bits)
N/A	`char ch = 'A';`	0 1 0 0 0 0 0 1
	`fprintf(fp, "A");`	
	`char ch = 'A';` `fprintf(fp, "%c", ch);`	
	`fprintf(fp, "5");`	
	`char ch = '5';` `fprintf(fp,"%c",ch);`	
	`int i = 5;`	

(Assuming size of an integer is 4 bytes)

IO Worksheet

What's written to the file? (show the bits)	Code	What's in the Memory? (show the bits)
N/A	`char ch = 'A';`	0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1	`fprintf(fp, "A");`	N/A
	`char ch = 'A';` `fprintf(fp, "%c", ch);`	
	`fprintf(fp, "5");`	
	`char ch = '5';` `fprintf(fp,"%c",ch);`	
	`int i = 5;`	

(Assuming size of an integer is 4 bytes)

IO Worksheet

What's written to the file? (show the bits)	**Code**	**What's in the Memory?** (show the bits)
N/A	`char ch = 'A';`	0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1	`fprintf(fp, "A");`	N/A
0 1 0 0 0 0 0 1	`char ch = 'A';` `fprintf(fp, "%c", ch);`	0 1 0 0 0 0 0 1
	`fprintf(fp, "5");`	
	`char ch = '5';` `fprintf(fp,"%c",ch);`	
	`int i = 5;`	

(Assuming size of an integer is 4 bytes)

IO Worksheet

What's written to the file? (show the bits)	**Code**	**What's in the Memory?** (show the bits)
N/A	`char ch = 'A';`	0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1	`fprintf(fp, "A");`	N/A
0 1 0 0 0 0 0 1	`char ch = 'A';` `fprintf(fp, "%c", ch);`	0 1 0 0 0 0 0 1
0 0 1 1 0 1 0 1	`fprintf(fp, "5");`	N/A
	`char ch = '5';` `fprintf(fp,"%c",ch);`	
	`int i = 5;`	

(Assuming size of an integer is 4 bytes)

IO Worksheet

What's written to the file? (show the bits)	Code	What's in the Memory? (show the bits)
N/A	`char ch = 'A';`	0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1	`fprintf(fp, "A");`	N/A
0 1 0 0 0 0 0 1	`char ch = 'A';` `fprintf(fp, "%c", ch);`	0 1 0 0 0 0 0 1
0 0 1 1 0 1 0 1	`fprintf(fp, "5");`	N/A
0 0 1 1 0 1 0 1	`char ch = '5';` `fprintf(fp,"%c",ch);`	0 0 1 1 0 1 0 1
	`int i = 5;`	

(Assuming size of an integer is 4 bytes)

IO Worksheet

What's written to the file? (show the bits)	Code	What's in the Memory? (show the bits)
N/A	`char ch = 'A';`	0 1 0 0 0 0 0 1
0 1 0 0 0 0 0 1	`fprintf(fp, "A");`	N/A
0 1 0 0 0 0 0 1	`char ch = 'A';` `fprintf(fp, "%c", ch);`	0 1 0 0 0 0 0 1
0 0 1 1 0 1 0 1	`fprintf(fp, "5");`	N/A
0 0 1 1 0 1 0 1	`char ch = '5';` `fprintf(fp,"%c",ch);`	0 0 1 1 0 1 0 1
	`int i = 5;`	00000000 00000000 00000000 00000101

(Assuming size of an integer is 4 bytes)

IO Worksheet

What's written to the file? (show the bits)	**Code**	**What's in the Memory?** (show the bits)
	`int i = 5;` `fprintf(fp,"%d",i);`	
	`int i = 5;` `fwrite(&i, sizeof(int), 1, fp);`	
	`char ch = '5';` `fwrite(&ch, sizeof(char), 1, fp);`	

Observe that text representation of 5 takes one byte, binary representation of integer takes 4 bytes.

What's written to the file? (show the bits)	Code	What's in the Memory? (show the bits)
0 0 1 1 0 1 0 1	`int i = 5;` `fprintf(fp,"%d",i);`	00000000 00000000 00000000 00000101
	`int i = 5;` `fwrite(&i, sizeof(int), 1, fp);`	
	`char ch = '5';` `fwrite(&ch, sizeof(char), 1, fp);`	

Observe that text representation of 5 takes one byte, binary representation of integer takes 4 bytes.

IO Worksheet

What's written to the file? (show the bits)	**Code**	**What's in the Memory?** (show the bits)
0 0 1 1 0 1 0 1	`int i = 5;` `fprintf(fp,"%d",i);`	00000000 00000000 00000000 00000101
00000000 00000000 00000000 00000101	`int i = 5;` `fwrite(&i, sizeof(int), 1, fp);`	00000000 00000000 00000000 00000101
	`char ch = ’5’;` `fwrite(&ch, sizeof(char), 1, fp);`	

Observe that text representation of 5 takes one byte, binary representation of integer takes 4 bytes.

IO Worksheet

What's written to the file? (show the bits)	**Code**	**What's in the Memory?** (show the bits)
0 0 1 1 0 1 0 1	`int i = 5;` `fprintf(fp,"%d",i);`	00000000 00000000 00000000 00000101
00000000 00000000 00000000 00000101	`int i = 5;` `fwrite(&i, sizeof(int), 1, fp);`	00000000 00000000 00000000 00000101
0 0 1 1 0 1 0 1	`char ch = '5';` `fwrite(&ch, sizeof(char), 1, fp);`	0 0 1 1 0 1 0 1

Observe that text representation of 5 takes one byte, binary representation of integer takes 4 bytes.

Binary versus text representation of integers

- Binary representation of integers is 4 bytes (assuming 32-bit architecture)
- Number of characters in text representation is equal to number of decimal digits. E.g. integer 999 takes 3 characters.

Binary versus text representation of integers

- Binary representation of integers is 4 bytes (assuming 32-bit architecture)
- Number of characters in text representation is equal to number of decimal digits. E.g. integer 999 takes 3 characters.

What range of integers uses more bytes to store in binary than text?

$\underbrace{ddd...ddd}_{\text{n } digits} \rightsquigarrow n$ bytes in text/char $\Rightarrow n > 4$

You should have noticed that more bytes are written for the `int i` than just a single character. Identify (with justification) the range of integers `i` such that:

1. more bytes are required to store `i` in binary –using `fwrite(&i, sizeof(int), 1, fp)`– than in text –using `fprintf(fp, "%d", i)`–.
2. the same number of bytes are required to store `i` in binary and in text.
3. fewer bytes are required to store `i` in binary than in text.

You should have noticed that more bytes are written for the `int i` than just a single character. Identify (with justification) the range of integers `i` such that:

1. more bytes are required to store `i` in binary –using `fwrite(&i, sizeof(int), 1, fp)`– than in text –using `fprintf(fp, "%d", i)`–.
   0-999
2. the same number of bytes are required to store `i` in binary and in text.
3. fewer bytes are required to store `i` in binary than in text.

You should have noticed that more bytes are written for the `int i` than just a single character. Identify (with justification) the range of integers `i` such that:

1. more bytes are required to store `i` in binary –using `fwrite(&i, sizeof(int), 1, fp)`– than in text –using `fprintf(fp, "%d", i)`–.
   0-999
2. the same number of bytes are required to store `i` in binary and in text.
   1000-9999
3. fewer bytes are required to store `i` in binary than in text.

You should have noticed that more bytes are written for the `int i` than just a single character. Identify (with justification) the range of integers `i` such that:

1. more bytes are required to store `i` in binary –using `fwrite(&i, sizeof(int), 1, fp)`– than in text –using `fprintf(fp, "%d", i)`–.
   0-999
2. the same number of bytes are required to store `i` in binary and in text.
   1000-9999
3. fewer bytes are required to store `i` in binary than in text.
   $>=$ 10000