Structures in Racket vs. C

Structures in C are similar to structures in Racket:

Racket:
(struct posn (x y))
(define p (posn 1 2))
(define a (posn-x p))
(define b (posn-y p))

C:
struct posn {
  int x;

  int y;
};

const struct posn p = {1,2};
const int a = p.x;
const int b = p.y;

Racket generates selector functions when you define a structure.
Instead of selector functions, C has a structure operator (.)
which selects the value of the requested field.

Call Stack and Stack Frames

Suppose the function main calls f, then f calls g, and g calls h.

As the program jumps from function to function, we need to
remember the history of the return addresses, as well as all the
parameters and local variables. This history is known as the call
stack.

The entries that are pushed onto the call stack are known as stack
frames.
Each function call creates a new stack frame that contains the following:

```plaintext
// ===========================
// <function name>:
// <parameter one>: <value>
// <parameter two>: <value>
// ...
// <local variable one>: ( <value> || ??? )
// <local variable two>: ( <value> || ??? )
// ...
// return address: <caller function name>:<line>
// ===========================
```

If a question asks you to draw the call stack, then for each stack frame pushed onto the call stack you must provide the above.

For example:

```c
int g(void){
    int x = 2;
    return x;
}

int f(int x){
    int a = g();
    return x * a;
}

int main(void){
    int x = 4;
    int a = 1;
    int y = f(x) + a;
}
```

Example: Draw the call stack

```plaintext
main:
    x: 4
    a: 1
    y: ?
    return address: OS
```

```c
int g(void){
    int x = 2;
    return x;
}

int f(int x){
    int a = g();
    return x * a;
}

int main(void){
    int x = 4;
    int a = 1;
    int y = f(x) + a;
}
```
Example: Draw the call stack

```c
int g(void){
    int x = 2;
    return x;
}

int f(int x){
    int a = g();
    return x * a;
}

int main(void){
    int x = 4;
    int a = 1;
    int y = f(x) + a;
}
```

CS 136 Spring 2019 Tutorial 5 7

Example: Draw the call stack

```c
int g(void){
    int x = 2;
    return x;
}

int f(int x){
    int a = g();
    return x * a;
}

int main(void){
    int x = 4;
    int a = 1;
    int y = f(x) + a;
}
```

CS 136 Spring 2019 Tutorial 5 8

Example: Draw the call stack

```c
int g(void){
    int x = 2;
    return x;
}

int f(int x){
    int a = g();
    return x * a;
}

int main(void){
    int x = 4;
    int a = 1;
    int y = f(x) + a;
}
```

CS 136 Spring 2019 Tutorial 5 9
Example: Draw the call stack

```c
int g(void){
    int x = 2;
    return x;
}

int f(int x){
    int a = g();
    return x * a;
}

int main(void){
    int x = 4;
    int a = 1;
    ⇒ int y = f(x) + a;

    =========================
    main:
    x: 4
    a: 1
    y: 9
    return address: OS
    =========================
}
```

Structures on the Stack

- Structures are just groups of regular variables.
- Also stored on the stack in the order fields are declared.
- Passed by value (one reason pointers are useful).

For Example:

```c
struct foo {
    int x;
    int y;
};

int baz(struct foo qux) {
    return qux.x + 1;
}

int main(void) {
    struct foo bar = {5, 6};
    int x = baz(bar);
    return 0;
}
```
Example: Structures on the call stack

```c
struct foo {
    int x;
    int y;
};

int baz(struct foo qux) {
    return qux.x + 1;
}

int main(void) {
    struct foo bar = {5, 6};
    return 0;
}
```

Example: Structures on the call stack

```c
struct foo {
    int x;
    int y;
};

int baz(struct foo qux) {
    return qux.x + 1;
}

int main(void) {
    struct foo bar = {5, 6};
    return 0;
}
```

Example: Structures on the call stack

```c
struct foo {
    int x;
    int y;
};

int baz(struct foo qux) {
    return qux.x + 1;
}

int main(void) {
    struct foo bar = {5, 6};
    return 0;
}
```
Memory Addresses and Pointers

This is a pointer:

```c
int i = 42;
int *p = &i;
```

Note that a pointer in this example is just a variable named \( p \) of the type \texttt{int *}.

The type of a pointer is first the \texttt{type} of the variable it points to and then an \texttt{asterisk} \(*\).

You can get the address of where a variable is stored in memory by using the \texttt{address operator} (\&).

To get the value of what a pointer "points at", we use the \texttt{indirection operator} (*) . This is often known as the dereference operator and it is the \texttt{inverse} of the address operator (\&).

```c
// To print the value p is pointing at
printf("The value of *p is: %d", *p);

// To print the value of p
printf("The value of p is: %p", p);

// To print the address of p
printf("The address of p is: %p", &p);
```

Pointers as Function Parameters

- Allows functions to mutate variables that live outside the function.
- Allows functions to avoid copying large structures.
- Allows functions to "return" multiple values.
Example: Mutating Values in Functions

Compare these two segments of code

```c
void inc(int p) {
    p += 1;
}

int main(void) {
    int x = 42;
    inc(x);
    printf("%d\n", x);
}
```

Output: 42

```c
void inc(int * p) {
    *p += 1;
}

int main(void) {
    int x = 42;
    inc(&x);
    printf("%d\n", x);
}
```

Output: 43

Example: Call Stacks with Pointers

Trace the content of the Call Stack for the following program

```c
void swap_int(int *i, int *j) {
    int temp = *i;
    *i = *j;
    *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
    swap_int(&(p1->x), &(p2->x));
    swap_int(&(p1->y), &(p2->y));
}

int main(void) {
    struct posn p1 = {3, 4};
    struct posn p2 = {5, 6};
    swap_posn(&p1, &p2);
}
```

```
main:
p1:
    .x = 3
    .y = 4
p2:
    .x = 5
    .y = 6
return address: OS
```

Example: Call Stacks with Pointers
Example: Call Stacks with Pointers

```c
void swap_int(int *i, int *j) {
    int temp = *i;
    *i = *j;
    *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
    swap_int(&(p1->x), &(p2->x));
    swap_int(&(p1->y), &(p2->y));
}

int main(void) {
    struct posn p1 = {3, 4};
    struct posn p2 = {5, 6};
    swap_posn(&p1, &p2);
}
```

---

Example: Call Stacks with Pointers

```c
void swap_int(int *i, int *j) {
    int temp = *i;
    *i = *j;
    *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
    swap_int(&(p1->x), &(p2->x));
    swap_int(&(p1->y), &(p2->y));
}

int main(void) {
    struct posn p1 = {3, 4};
    struct posn p2 = {5, 6};
    swap_posn(&p1, &p2);
}
```
Example: Call Stacks with Pointers

void swap_int(int *i, int *j) {
    int temp = *i;
    *i = *j;
    *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
    swap_int(&(p1->x), &(p2->x));
    swap_int(&(p1->y), &(p2->y));
}

int main(void) {
    struct posn p1 = {3, 4};
    struct posn p2 = {5, 6};
    swap_posn(&p1, &p2);
}

CS 136 Spring 2019 Tutorial 5 25
Example: Call Stacks with Pointers

```c
void swap_int(int *i, int *j) {
  int temp = *i;
  *i = *j;
  *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
  swap_int(&(p1->x), &(p2->x));
  swap_int(&(p1->y), &(p2->y));
}

int main(void) {
  struct posn p1 = {3, 4};
  struct posn p2 = {5, 6};
  swap_posn(&p1, &p2);
}
```

---

Example: Call Stacks with Pointers

```c
void swap_int(int *i, int *j) {
  int temp = *i;
  *i = *j;
  *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
  swap_int(&(p1->x), &(p2->x));
  swap_int(&(p1->y), &(p2->y));
}

int main(void) {
  struct posn p1 = {3, 4};
  struct posn p2 = {5, 6};
  swap_posn(&p1, &p2);
}
```
Example: Call Stacks with Pointers

```c
void swap_int(int *i, int *j) {
    int temp = *i;
    *i = *j;
    *j = temp;
}

void swap_posn(struct posn *p1, struct posn *p2) {
    swap_int(&(p1->x), &(p2->x));
    swap_int(&(p1->y), &(p2->y));
}

int main(void) {
    struct posn p1 = {3, 4};
    struct posn p2 = {5, 6};
    swap_posn(&p1, &p2);
}
```

End of Stack Trace