C

CS 045

Computer Organization and Architecture

Prof. Donald J. Patterson

TYPES IN C

STRUCT

- The forerunner of the class

```c
#include <stdio.h>

int main(){

    struct protoclass {
        int a;
        int b;
        char c;
        float d;
    } e;

    e.a = 1;
    e.b = 2;
    e.c = 'x';
    e.d = 2.7;

    printf("struct                    size: %lu\n",sizeof(e));

    return 0;
}
```

TYPES IN C

STRUCT

- The forerunner of the class

```
struct                          size: 16
```

```c
#include <stdio.h>

int main(){

    struct protoclass {
        int a;
        int b;
        char c;
        float d;
    } e;

    e.a = 1;
    e.b = 2;
    e.c = 'x';
    e.d = 2.7;

    printf("struct                  size: %lu\n",sizeof(e));

    return 0;
}
```

TYPES IN C

ARRAY

- aka, List

```c
#include <stdio.h>

int main(){

    int a[10];
    int b[5];
    double c[2];

    a[0]=0;
    a[1]=1;
    a[2]=4;
    a[3]=6;
    a[4]=8;
    a[5]=10;
    a[6]=12;
    a[7]=14;
    a[8]=16;
    a[9]=18;

    c[0] = 0.0;

    printf("sizeof(a) is %lu\n",sizeof(a));
    printf("sizeof(b) is %lu\n",sizeof(b));
    printf("sizeof(c) is %lu\n",sizeof(c));
    printf("sizeof(a[0]) is %lu\n",sizeof(a[0]));
    printf("sizeof(b[0]) is %lu\n",sizeof(b[0]));
    printf("sizeof(c[0]) is %lu\n",sizeof(c[0]));

    printf("a: %p\n",a);
    printf("&(a[0]): %p\n",&(a[0]));
    printf("&(a[1]): %p\n",&(a[1]));


    return 0;
}
```

TYPES IN C

ARRAY

- aka, List

```c
#include <stdio.h>

int main(){

    int a[10];
    int b[5];
    double c[2];

    a[0]=0;
    a[1]=1;
    a[2]=4;
    a[3]=6;
    a[4]=8;
    a[5]=10;
    a[6]=12;
    a[7]=14;
    a[8]=16;
    a[9]=18;

    c[0] = 0.0;

    printf("sizeof(a) is %lu\n",sizeof(a));
    printf("sizeof(b) is %lu\n",sizeof(b));
    printf("sizeof(c) is %lu\n",sizeof(c));
    printf("sizeof(a[0]) is %lu\n",sizeof(a[0]));
    printf("sizeof(b[0]) is %lu\n",sizeof(b[0]));
    printf("sizeof(c[0]) is %lu\n",sizeof(c[0]));

    printf("a: %p\n",a);
    printf("&(a[0]): %p\n",&(a[0]));
    printf("&(a[1]): %p\n",&(a[1]));


    return 0;
}
```

```
sizeof(a) is 40
sizeof(b) is 20
sizeof(c) is 16
sizeof(a[0]) is 4
sizeof(b[0]) is 4
sizeof(c[0]) is 8
a: 0x7fff57052340
&(a[0]): 0x7fff57052340
&(a[1]): 0x7fff57052344
```

TYPES IN C

MULTI-DIMENSIONAL ARRAY

- aka, Matrix

```c
#include <stdio.h>
int main(){
    int a[2];
    int b[2][3];
    double c[5][5][5];

    a[0] = 0;
    b[1][1] = 1;
    c[4][3][1] = 1;

    printf("sizeof(a) is %lu\n",sizeof(a));
    printf("sizeof(b) is %lu\n",sizeof(b));
    printf("sizeof(c) is %lu\n",sizeof(c));
    printf("sizeof(a[0]) is %lu\n",sizeof(a[0]));
    printf("sizeof(b[1][1]) is %lu\n",sizeof(b[1][1]));
    printf("sizeof(c[4][3][1]) is %lu\n",sizeof(c[4][3][1]));
    printf("sizeof(c[1][1][2]) is %lu\n",sizeof(c[1][1][2]));

    return 0;
}
```

TYPES IN C

MULTI-DIMENSIONAL ARRAY

- aka, Matrix

```
sizeof(a) is 8
sizeof(b) is 24
sizeof(c) is 1000
sizeof(a[0]) is 4
sizeof(b[1][1]) is 4
sizeof(c[4][3][1]) is 8
sizeof(c[1][1][2]) is 8
```

```c
#include <stdio.h>
int main(){

    int a[2];
    int b[2][3];
    double c[5][5][5];

    a[0] = 0;
    b[1][1] = 1;
    c[4][3][1] = 1;

    printf("sizeof(a) is %lu\n",sizeof(a));
    printf("sizeof(b) is %lu\n",sizeof(b));
    printf("sizeof(c) is %lu\n",sizeof(c));
    printf("sizeof(a[0]) is %lu\n",sizeof(a[0]));
    printf("sizeof(b[1][1]) is %lu\n",sizeof(b[1][1]));
    printf("sizeof(c[4][3][1]) is %lu\n",sizeof(c[4][3][1]));
    printf("sizeof(c[1][1][2]) is %lu\n",sizeof(c[1][1][2]));

    return 0;
}
```

STRINGS IN C

- An array of chars that end with a zero
  - Shorthand for such an array is “<string>”

```c
#include <stdio.h>

int main () {

   char name_a[10] = {'P', 'a', 't', 't', 'e', 'r', 's', 'o', 'n', '\0'};
   char name_b[10] = "Patterson";

   printf("Name_a: %s\n", name_a );
   printf("Name_b: %s\n", name_b );

   return 0;
}
```

STRINGS IN C

- An array of chars that end with a zero
  - Shorthand for such an array is “<string>”

```
Name_a: Patterson
Name_b: Patterson
```

```c
#include <stdio.h>

int main () {

   char name_a[10] = {'P', 'a', 't', 't', 'e', 'r', 's', 'o', 'n', '\0'};
   char name_b[10] = "Patterson";

   printf("Name_a: %s\n", name_a );
   printf("Name_b: %s\n", name_b );

   return 0;
}
```

CONTROL STRUCTURES

IF-ELSE STATEMENT

```c
#include <stdio.h>

int main(){

    int x = 5;

    if ( x < 10 ){
        printf("x is less than 10\n");
    } else {
        printf("x is greater than 10\n");
    }

    if ( x == 10 ){
        printf("x equals 10\n");
    } else {
        printf("x does not equal 10\n");
    }


    return 0;
}
```

CONTROL STRUCTURES

IF-ELSE STATEMENT

```c
#include <stdio.h>

int main(){

    int x = 5;

    if ( x < 10 ){
        printf("x is less than 10\n");
    } else {
        printf("x is greater than 10\n");
    }

    if ( x == 10 ){
        printf("x equals 10\n");
    } else {
        printf("x does not equal 10\n");
    }


    return 0;
}
```

```
x is less than 10
x does not equal 10
```

CONTROL STRUCTURES

SWITCH STATEMENT

- shorthand for a sequence of if statements

```c
#include <stdio.h>

int main(){

    int x = 5;

    switch(x) {
        case 1: printf("x is 1\n");
        case 2: printf("x is 1\n");
        case 3: printf("x is 1\n");
        case 4: printf("x is 1\n");
        case 5: printf("x is 1\n");
        default:printf("x is less than 1 or greater than 5\n");
    }

    int y = 4;

    switch(x) {
        case 1: printf("y is 1\n");
                break;
        case 2: printf("y is 2\n");
                break;
        case 3: printf("y is 3\n");
                break;
        case 4: printf("y is 4\n");
                break;
        case 5: printf("y is 5\n");
                break;
        default:printf("y is less than 1 or greater than 5\n");
    }


    return 0;
}
```

CONTROL STRUCTURES

SWITCH STATEMENT

- shorthand for a sequence of if statements

```
x is 1
x is less than 1 or greater than 5
y is 5
```

```c
#include <stdio.h>

int main(){

    int x = 5;

    switch(x) {
        case 1: printf("x is 1\n");
        case 2: printf("x is 1\n");
        case 3: printf("x is 1\n");
        case 4: printf("x is 1\n");
        case 5: printf("x is 1\n");
        default:printf("x is less than 1 or greater than 5\n");
    }

    int y = 4;

    switch(x) {
        case 1: printf("y is 1\n");
                break;
        case 2: printf("y is 2\n");
                break;
        case 3: printf("y is 3\n");
                break;
        case 4: printf("y is 4\n");
                break;
        case 5: printf("y is 5\n");
                break;
        default:printf("y is less than 1 or greater than 5\n");
    }


    return 0;
}
```

LOOPS

WHILE

```c
#include <stdio.h>

int main(){

    int x = 5;

    while( x >= 0 ){
        printf("x is %d\n",x);

        //x = x - 1
        x--;
    }

    return 0;
}
```

LOOPS

WHILE

```c
#include <stdio.h>

int main(){

    int x = 5;

    while( x >= 0 ){
        printf("x is %d\n",x);

        //x = x - 1
        x--;
    }

    return 0;
}
```

```
x is 5
x is 4
x is 3
x is 2
x is 1
x is 0
```

LOOPS

FOR

```c
#include <stdio.h>

int main(){

    int x = 5;

    for(x = 0; x < 5; x++){
        printf("x is %d\n",x);
    }

    return 0;
}
```

LOOPS

FOR

```c
#include <stdio.h>

int main(){

    int x = 5;

    for(x = 0; x < 5; x++){
        printf("x is %d\n",x);
    }

    return 0;
}
```

```
x is 0
x is 1
x is 2
x is 3
x is 4
```

FUNCTIONS

```c
#include <stdio.h>

void greeting(char timeOfDay[]){
    printf("Good %s\n",timeOfDay);
}



int main(){

    greeting("morning");
    greeting("afternoon");
    greeting("evening");

    return 0;
}
```

FUNCTIONS

```c
#include <stdio.h>

void greeting(char timeOfDay[]){

    printf("Good %s\n",timeOfDay);

}




int main(){

    greeting("morning");
    greeting("afternoon");
    greeting("evening");

    return 0;
}
```

```
Good morning
Good afternoon
Good evening
```

FUNCTIONS

```c
#include <stdio.h>

int doubleMe(int x){

    x = x + x;
    return x;

}



int main(){

    int x = 2;
    printf("x is %d\n",x);
    printf("When you double %d you get %d\n",x,doubleMe(x));
    printf("x is %d\n",x);
    printf("\n");

    x = 5;
    printf("x is %d\n",x);
    printf("When you double %d you get %d\n",x,doubleMe(x));
    printf("x is %d\n",x);
    printf("\n");

    x = 100;
    printf("x is %d\n",x);
    printf("When you double %d you get %d\n",x,doubleMe(x));
    printf("x is %d\n",x);
    printf("\n");

    return 0;
}
```

FUNCTIONS

```c
#include <stdio.h>

int doubleMe(int x){

    x = x + x;
    return x;

}



int main(){

    int x = 2;
    printf("x is %d\n",x);
    printf("When you double %d you get %d\n",x,doubleMe(x));
    printf("x is %d\n",x);
    printf("\n");

    x = 5;
    printf("x is %d\n",x);
    printf("When you double %d you get %d\n",x,doubleMe(x));
    printf("x is %d\n",x);
    printf("\n");

    x = 100;
    printf("x is %d\n",x);
    printf("When you double %d you get %d\n",x,doubleMe(x));
    printf("x is %d\n",x);
    printf("\n");

    return 0;
}
```

```
x is 2
When you double 2 you get 4
x is 2

x is 5
When you double 5 you get 10
x is 5

x is 100
When you double 100 you get 200
x is 100
```

TYPES IN C : POINTERS

MEMORY

- C programs actually have two kinds of memory

- One kind of memory is for frames and local variables - that's stack memory

- When one makes a recursive program that calls itself forever, one runs out of stack memory

TYPES IN C : POINTERS

MEMORY

- The second kind of memory is heap memory

- You can think of memory as a long list of 0's and 1's

0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	1	1	0	1	1	0	1	0	1	1	0	1	1	0	0	0	1	0	1	1	0	1	1

TYPES IN C : POINTERS

MEMORY

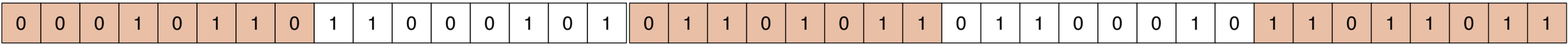
- The 0's and 1's are chunked into bytes of 8 bits

0	0	0	1	0	1	1	0	1	1	0	0	0	1	0	1	0	1	1	0	1	0	1	1	0	1	1	0	0	0	1	0	1	1	0	1	1	0	1	1

TYPES IN C : POINTERS

MEMORY

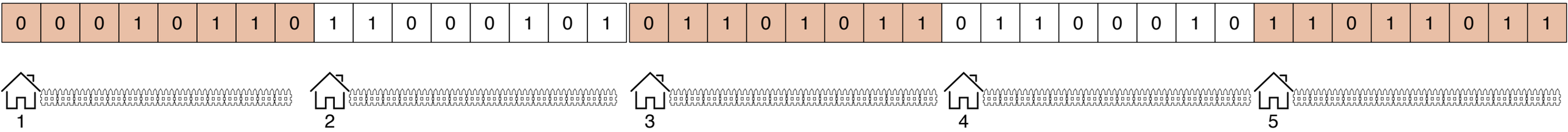
- Each byte is given an address
  - Similar to a street address for a house
  - So that the computer can find the byte



TYPES IN C : POINTERS

MEMORY

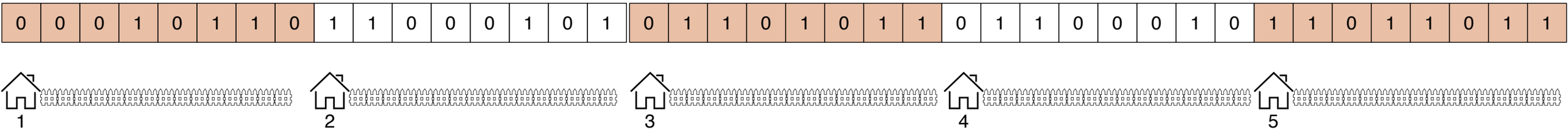
- Each byte is given an address
  - Similar to a street address for a house
  - So that the computer can find the byte



TYPES IN C : POINTERS

MEMORY

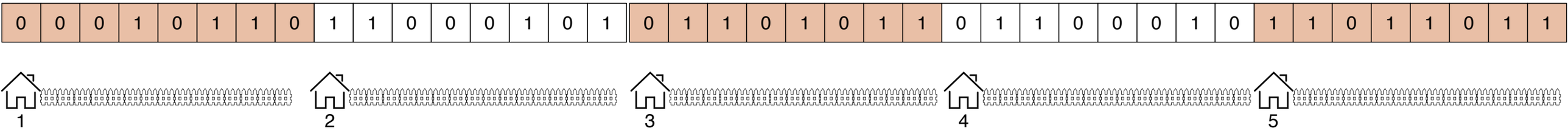
- If x is a 16-bit integer, it uses 2 bytes



TYPES IN C : POINTERS

MEMORY

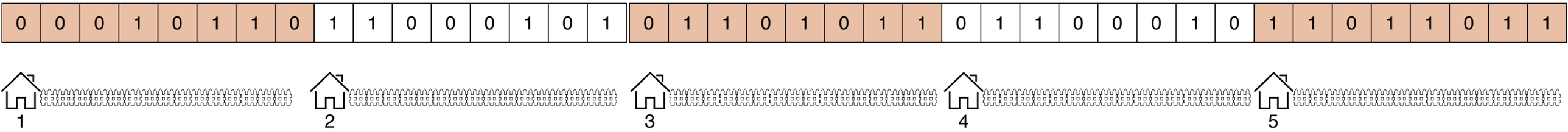
- if you make a function call, then a copy is made on the stack



TYPES IN C : POINTERS

MEMORY

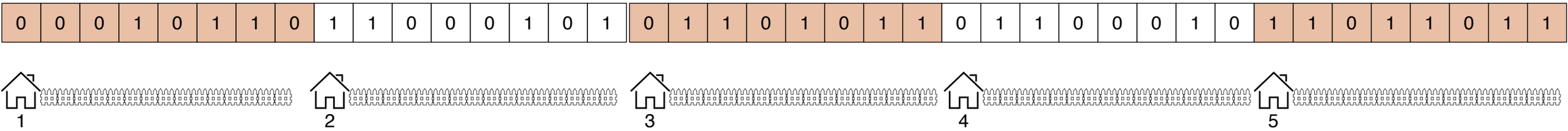
- if you make a function call, then a copy is made on the stack



TYPES IN C : POINTERS

MEMORY

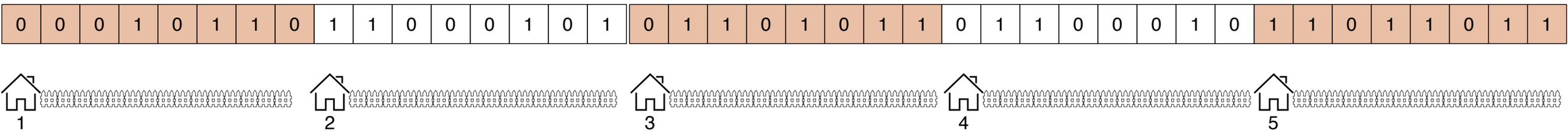
- And when the function returns it is removed



TYPES IN C : POINTERS

MEMORY

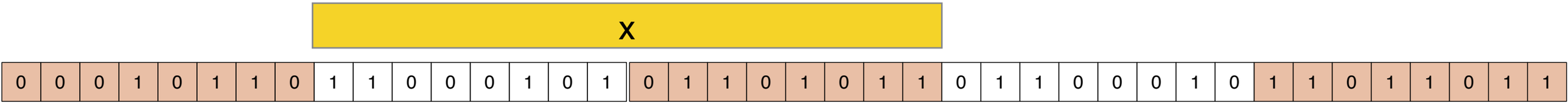
- And when the function returns it is removed



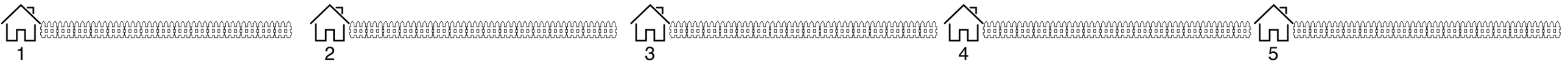
TYPES IN C : POINTERS

MEMORY

- For function calls and method calls, this copying and destroying is handled automatically



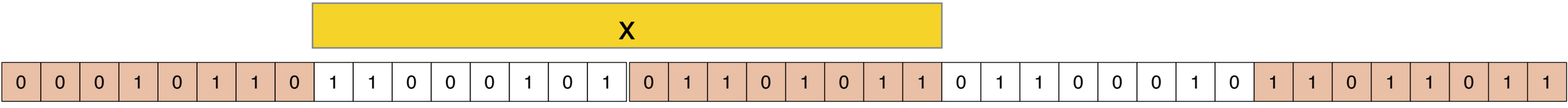
- on stack memory

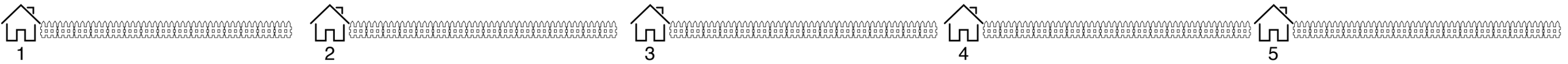


TYPES IN C : POINTERS

MEMORY

- If a programmer wants changes to x to persist, then she has to manage the memory herself



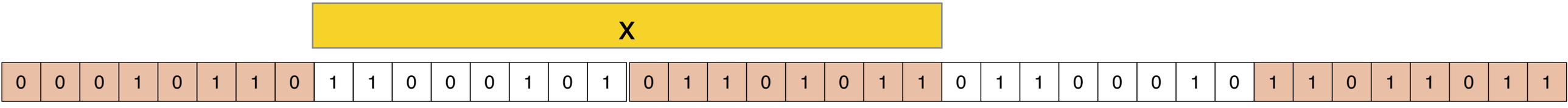


- and that requires using the address of x

TYPES IN C : POINTERS

MEMORY

- In this picture, the address of x is “2”
- 2 is a pointer to x



- If we want to keep track of pieces of memory use pointers

TYPES IN C : POINTERS

POINTERS

- We've seen how to declare an integer

```
int x;
```

- If we want to declare a pointer to an integer, we use the star notation

```
int *y;
```

- The type of this variable includes the star

```
int *
```

TYPES IN C : POINTERS

POINTERS

```c
#include <stdio.h>

int main(){

    int a = 5;
    int *pointerToA = &a;

    printf("The value of a is \t\t%d\n",a);
    printf("The sizeof(a) is \t\t%lu\n",sizeof(a));
    printf("The address of a is \t\t%p\n",&a);

    printf("The value of pointerToA is \t%p\n",pointerToA);
    printf("The sizeof(pointerToA) is \t%lu\n",sizeof(pointerToA));
    printf("The address of pointerToA is \t%p\n",&pointerToA);


    return 0;
}
```

TYPES IN C : POINTERS

POINTERS

```
The value of a is               5
The sizeof(a) is                4
The address of a is             0x7fff532f0348
The value of pointerToA is      0x7fff532f0348
The sizeof(pointerToA) is       8
The address of pointerToA is    0x7fff532f0340
```

```c
#include <stdio.h>

int main(){

    int a = 5;
    int *pointerToA = &a;

    printf("The value of a is \t\t%d\n",a);
    printf("The sizeof(a) is \t\t%lu\n",sizeof(a));
    printf("The address of a is \t\t%p\n",&a);

    printf("The value of pointerToA is \t%p\n",pointerToA);
    printf("The sizeof(pointerToA) is \t%lu\n",sizeof(pointerToA));
    printf("The address of pointerToA is \t%p\n",&pointerToA);


    return 0;
}
```

TYPES IN C : POINTERS

USING POINTERS

- To get the address of a variable use an ampersand
- To get the value that an address points to use a star
  - this is called “dereferencing”

```
int y = 5;
int *x = &y;
(*x) = 6;
printf("y = %d\n",y);
```

- “x points to y”

TYPES IN C : POINTERS

USING POINTERS

- To get the address of a variable use an ampersand
- To get the value that an address points to use a star
  - this is called “dereferencing”

```
int y = 5;
int *x = &y;
(*x) = 6;
printf("y = %d\n",y);
```

```
y = 6
```

- “x points to y”

TYPES IN C : POINTERS

SPECIAL ADDRESS

- NULL is a special value
  - it is address “0”
- You can set pointers to NULL and then use that in conditionals

```
int y = 5;
int *x = NULL;

if(x == NULL){
    x = &y;
}
(*x) = 6;
printf("y = %d\n",y);
```

TYPES IN C : POINTERS

SPECIAL ADDRESS

- NULL is a special value
  - it is address “0”
- You can set pointers to NULL and then use that in conditionals

```
int y = 5;
int *x = NULL;

if(x == NULL){
    x = &y;
}
(*x) = 6;
printf("y = %d\n",y);
```

y = 6

TYPES IN C : POINTERS

BE CAREFUL!

- Just because you have a pointer doesn't mean it points to something meaningful!
- You must assign it first

```c
#include <stdio.h>

int main(){

    int y = 5;
    int *x;

    (*x) = 6;
    printf("y = %d\n",y);

    return 0;
}
```

TYPES IN C : POINTERS

BE CAREFUL!

- Just because you have a pointer doesn't mean it points to something meaningful!
- You must assign it first

```c
#include <stdio.h>

int main(){

    int y = 5;
    int *x;

    (*x) = 6;
    printf("y = %d\n",y);

    return 0;
}
```

```
y = 5
```

TYPES IN C : POINTERS

BE CAREFUL!

- Just because you have a pointer doesn't mean it points to something meaningful!
- You must assign it first

```c
#include <stdio.h>

int main(){

    int y = 5;
    int *x;

    (*x) = 6;!!!!!
    printf("y = %d\n",y);

    return 0;
}
```

```
y = 5
```

TYPES IN C : POINTERS

BE CAREFUL!

- Just because you have a pointer doesn't mean it points to something meaningful!
- You must assign it first

```c
#include <stdio.h>

int main(){

    int y = 5;
    int *x;

    (*x) = 6;!!!!!
    printf("y = %d\n",y);

    return 0;
}
```

```
y = 5
```

TYPES IN C : POINTERS

BE CAREFUL!

- Don’t dereference a NULL pointer!

```c
#include <stdio.h>

int main(){

    int *x = NULL;

    (*x) = 6;

    return 0;
}
```

```
Segmentation fault: 11
```

TYPES IN C : POINTERS

YOU CAN HAVE A POINTER TO ANY KIND OF VARIABLE

- Using the star notation is how you define it

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

int main(){

    int *A;
    char *B;
    double *C;
    int32_t *D;
    struct time *E;

    return 0;
}
```

WHAT'S NEXT?

- Advanced programmers can use pointers to keep track of heap memory
  - heap memory doesn't get destroyed unless the programmer tells the computer to destroy it

TYPES IN C : POINTERS

SUMMARY

- pointers are a new type
  - You can have a pointer to any other type

- The don’t keep track of data
  - they keep track of the address of data
  - the “point” to the data

- pointers allow you to pass parameters to functions when you want changes to persist after the function is done and the frame is destroyed

- You can dereference a pointer to access the data

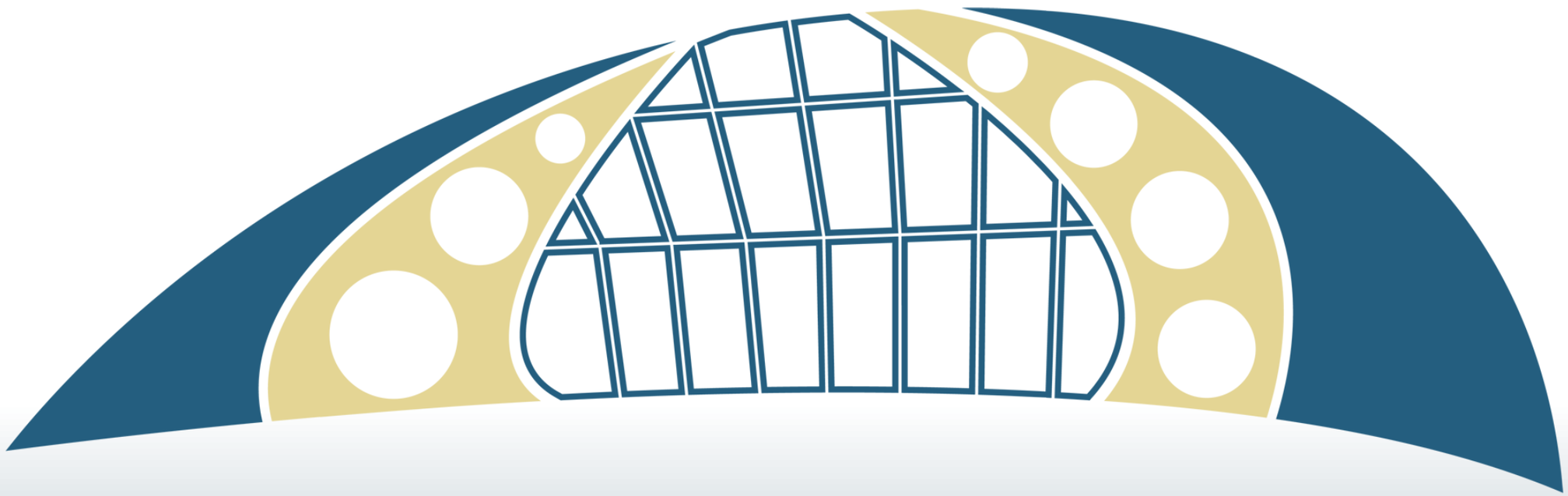
TYPES IN C : POINTERS

WORK IT OUT

- Create a program that will print out numbers from 1 to 100 backwards

- Create a program that
  - will create an array of integers with 100 elements in it
  - set each of the elements to twice its index value.
  - output the address of each element of the array

- Create a function that doubles the value of its argument. Use pointers to make the changes persist even after the function call is complete.



WESTMONT INSPIRED

COMPUTING LAB