Pointer Declarations

- Read from *right* to *left*

```
char   *cp;
int    *ip;
float  *fp;
int    **q;
struct *xp;
```

Pointer Declarations

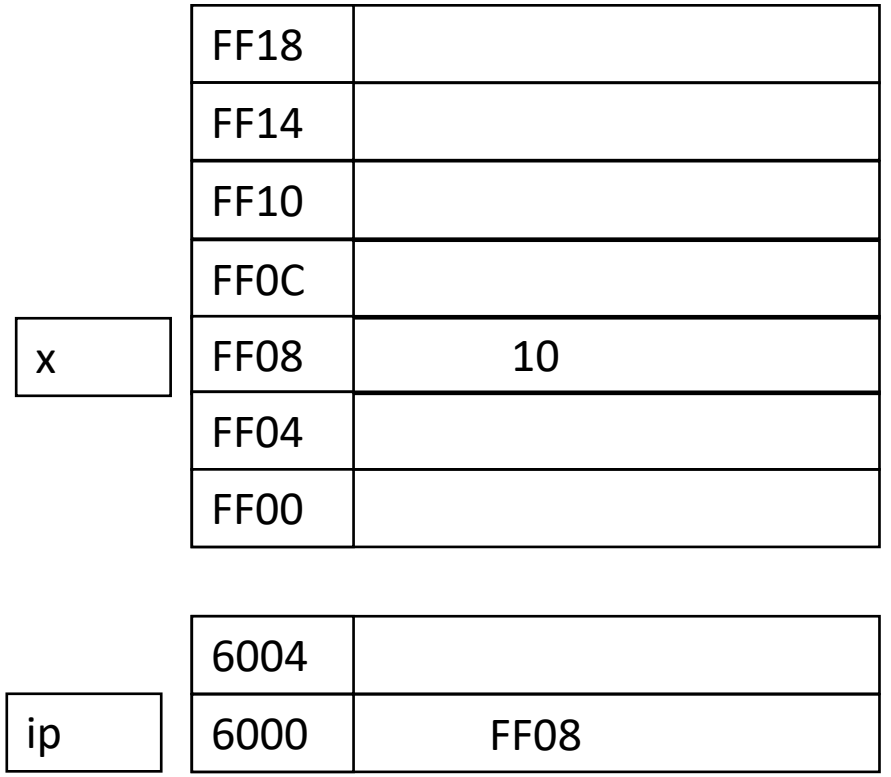
- Read from *right* to *left*

```
const int *ip; // pointer can change, thing it points to cannot
int * const q; // pointer cannot change, thing it points to can
const int * const r;
```

Using a Pointer

- Dereferencing

```
int *ip;
int  x = 10;
ip = &x;
```

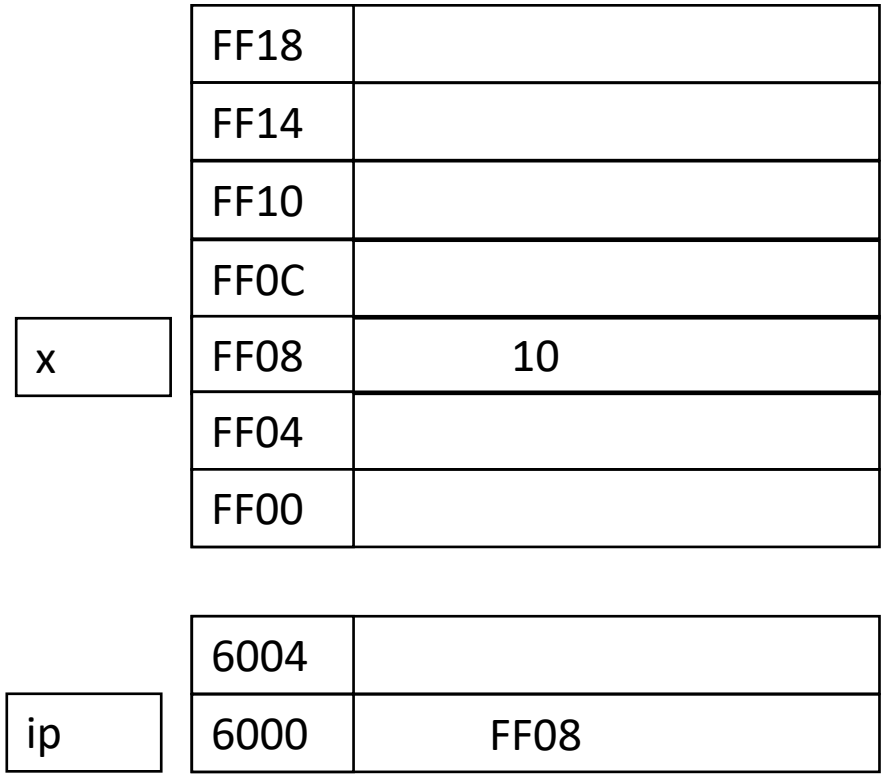


Using a Pointer

- Dereferencing

```
int *ip;
int  x = 10;
ip = &x;

printf (“%d\n”, *ip);
```

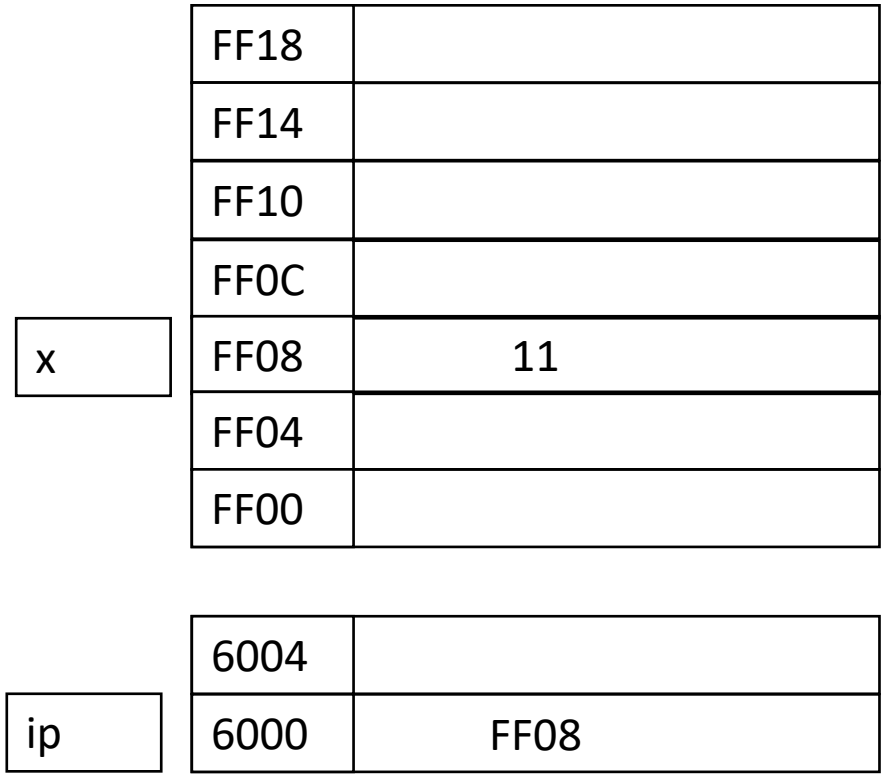


Using a Pointer

- Dereferencing

```
int *ip;
int  x = 10;
ip = &x;

printf (“%d\n”, *ip);
++*ip;
```

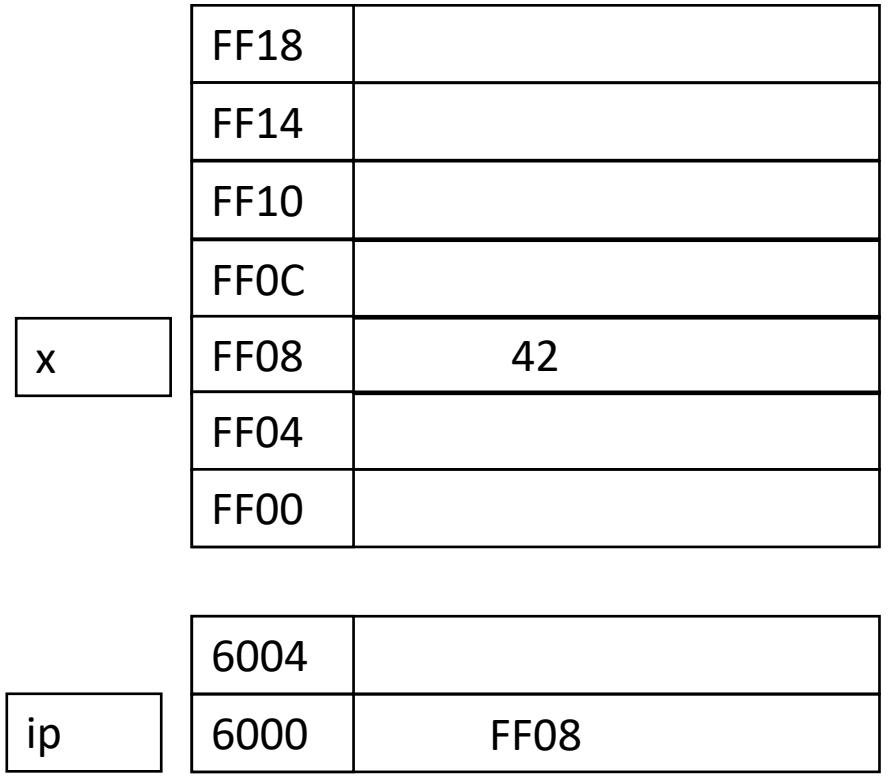


Using a Pointer

- Dereferencing

```
int *ip;
int  x = 10;
ip = &x;

printf (“%d\n”, *ip);
++*ip;
*ip = *ip * 4 - 2;
printf (“%d\n”, x);
```

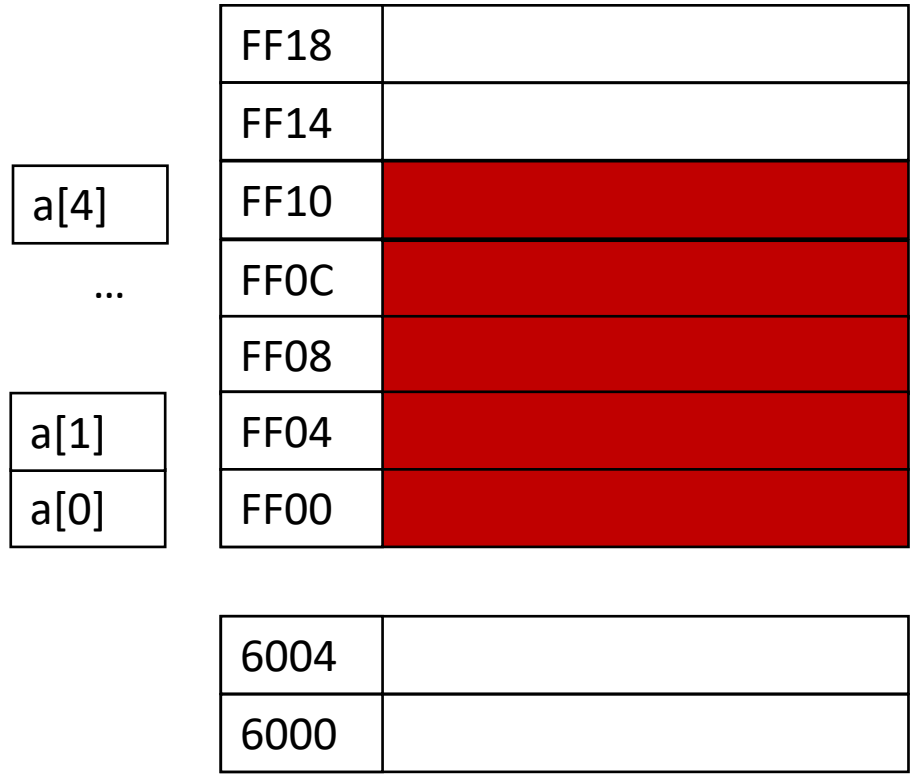


Demo

Pointers and Arrays

- Pointers and arrays are strongly related

```
int  a[5];
```

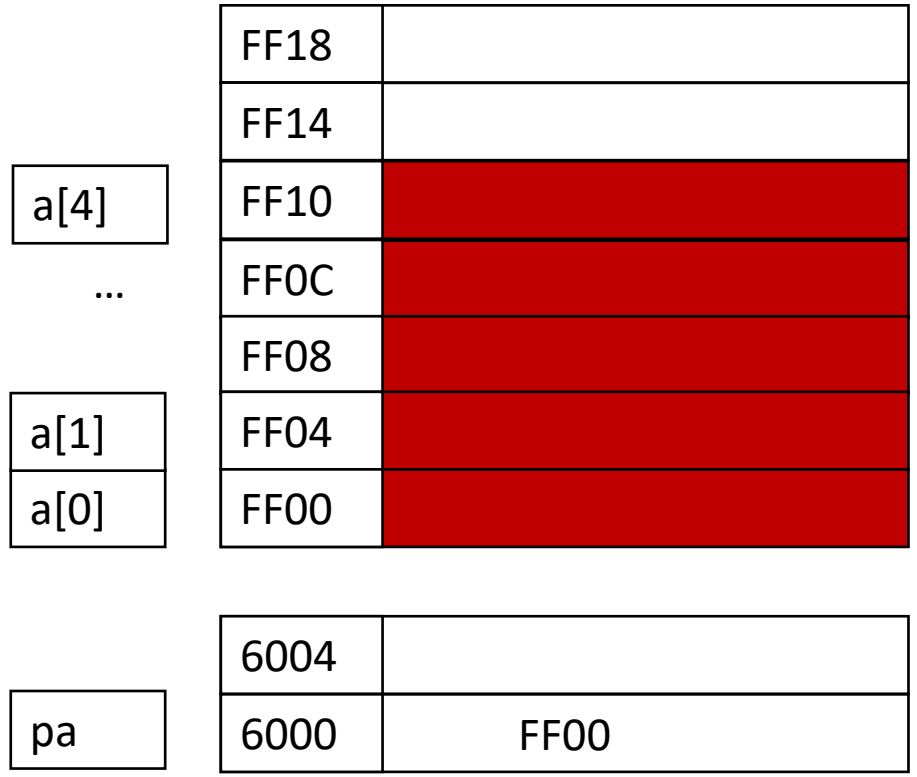


Pointers and Arrays

- Pointers and arrays are strongly related

```
int  a[5];
int  *pa;

pa = &a[0]
```

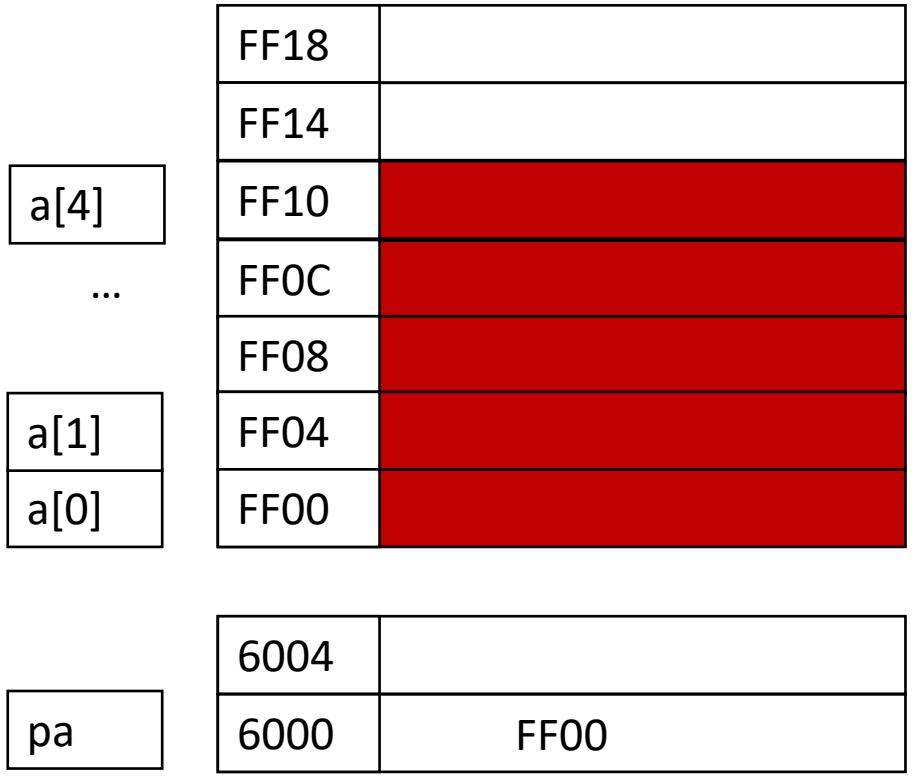


Pointers and Arrays

- Pointers and arrays are strongly related

```
int  a[5];
int  *pa;
pa = &a[0]

int x = *pa;
int y = *(pa+2); // y = a[2]
```



pa+i is the address of a[i], *(pa+i) is the contents of a[i]

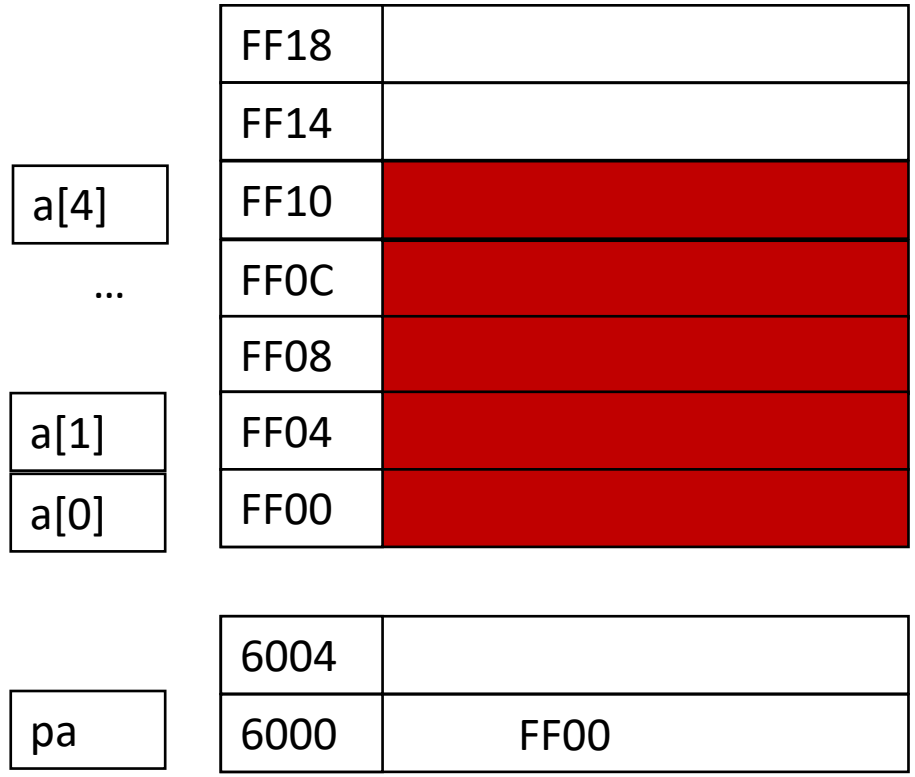
Pointers and Arrays

- Pointers and arrays are strongly related

```
int  a[5];
int  *pa;

pa = a; // same as before
```

`a[i]` can also be written as `*(a+i)`



Array-and-index expression is the same as pointer and offset expression

Pointers and Arrays

- `int a[10];` vs `int *a;`
  - **`int a[10]`** sets aside *ten* units of memory, and **`a`** is initialized to point to the zeroth unit
  - **`int *a`** sets aside *one* pointer-sized unit of memory, not initialized

Demo

Character Pointers for String Manipulation

- A string is an array of characters

```
char am[] = “hello”;
char *pm;
pm = &am;
```

FF06	
FF05	\0
FF04	o
FF03	l
FF02	l
FF01	e
FF00	h

	6004	
pm	6000	FF00

Character Pointers for String Manipulation

- A string is an array of characters

```
char am[] = “hello”;
char *pm;
pm = &am[0];

char *pm = “hello”
```

FF06	
FF05	\0
FF04	o
FF03	l
FF02	l
FF01	e
FF00	h

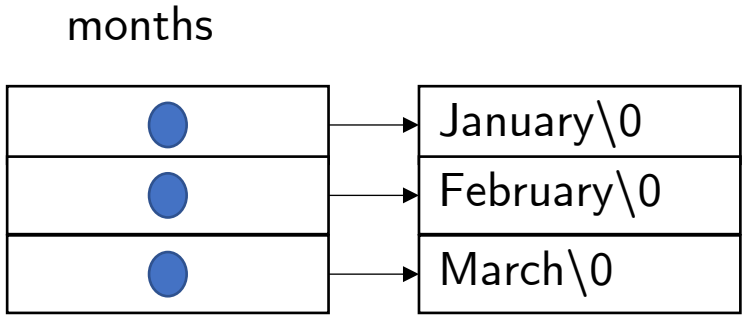
	6004	
pm	6000	FF00

Demo

Pointer Arrays

- Pointers are variable
- Pointer arrays are also possible

```
char *months[] = {"January",
                  "February",
                  "March" };
```



Pointer vs Multidimensional Arrays

- `int a[10][20] vs int *b[10]`

- `int a[10][20]` ➔ 200 locations set aside
- `int *b[10]` ➔ 10 locations set aside

- Each element of b can point need not to point same twenty-element vector

Demo

Address Arithmetic

- Pointers are variables

```
++ip;
iq = ip;
--ip;
printf (“%p”, ip);
```

- But, a pointer is constrained to a particular kind of object

Address Arithmetic

- Legal
  - Assignment of pointers of same type
  - Adding or subtracting a pointer to an integer
  - Subtracting or comparing two pointers (to members of the same array)
  - Assigning or comparing a pointer to zero
  - Increment or decrement a pointer using ++ or --.

All other operations are illegal

Demo

Pointers and Function Arguments

- C passes arguments to a function by value

Demo

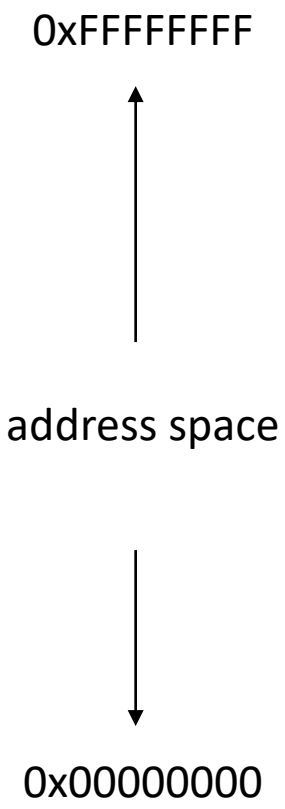
Memory Organization

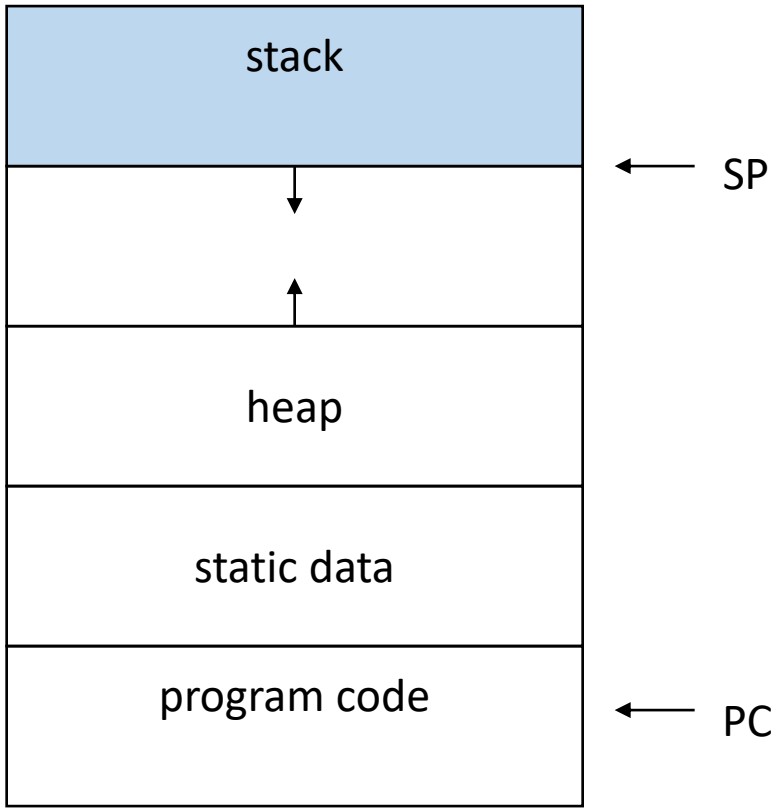
```
#include<stdio.h>

void increment(int i){
    i = i+1;
}

int main(){
    int  i = 10;
    increment(i);
    printf("i = %d\n",i);

    return 0;
}
```





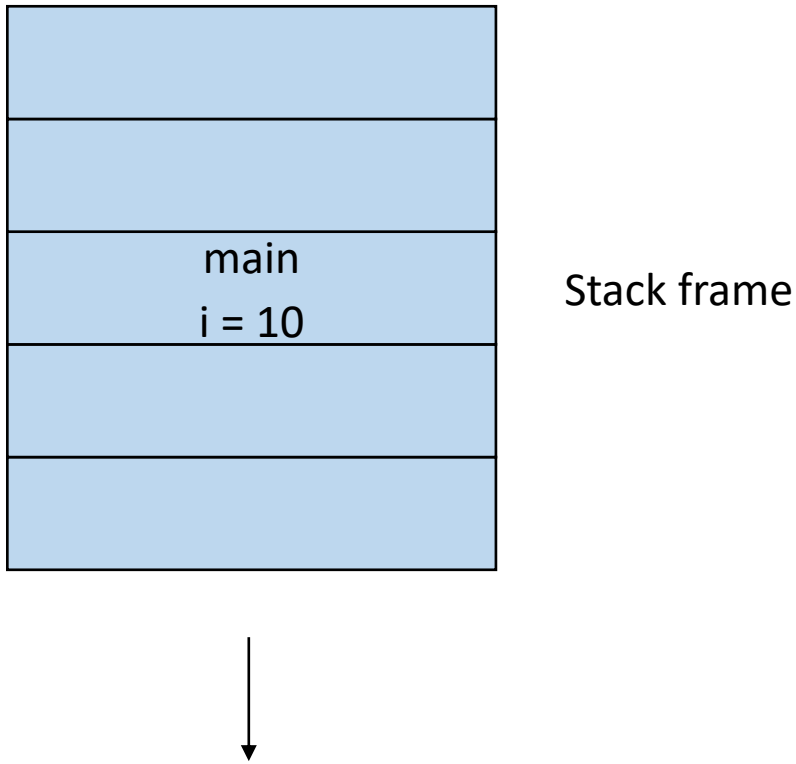
Memory Organization

```
    #include<stdio.h>

    void increment(int i){
        i = i+1;
    }

    int main(){
>>      int  i = 10;
        increment(i);
        printf("i = %d\n",i);

        return 0;
    }
```



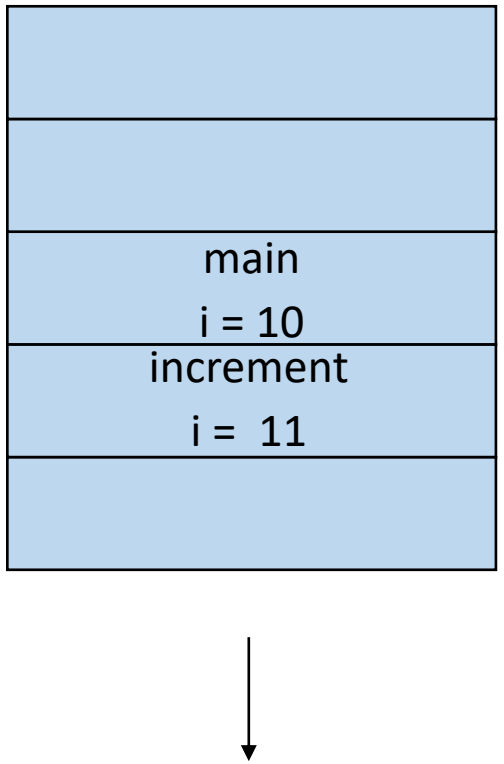
Memory Organization

```
      #include<stdio.h>

      void increment(int i){
>>        i = i+1;
      }

      int main(){
          int  i = 10;
          increment(i);
          printf("i = %d\n",i);

          return 0;
      }
```



Stack frame

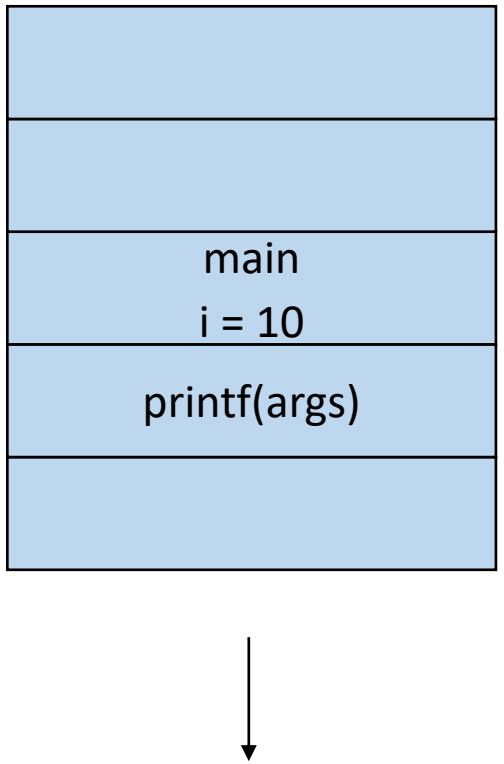
Memory Organization

```
#include<stdio.h>

void increment(int i){
    i = i+1;
}

int main(){
    int  i = 10;
    increment(i);
>>  printf("i = %d\n",i);

    return 0;
}
```



Stack frame

Pointers and Function Arguments

- Call by reference

```
#include<stdio.h>

void increment(int *i){
    (*i) = (*i)+1;
}

int main(){
    int  i = 10;
    increment(&i);
    printf("i = %d\n",i);

    return 0;
}
```

Demo

Pointers and Function Arguments

- Pass by reference -- No need to copy the data

- Command line arguments

```
int main(int argc, char *argv[])
<program name> arg0  arg1 arg2
```

Demo

Character Pointers & Functions

- Helpful in string manipulation

```
strlen(char *s);
strcpy(char *s, char *t);
strcmp(char *s, char *t)
```

Demo