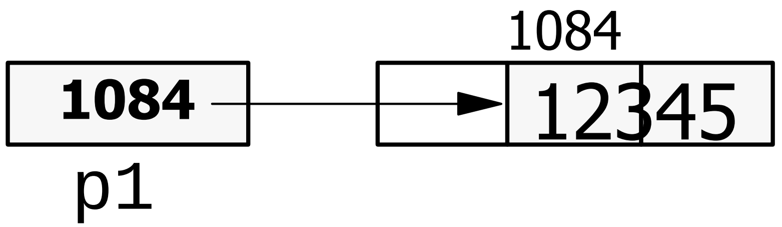
Dynamic Memory Allocation

- `malloc()` function
- accepts the number of bytes needed
- allocates a block of memory at run time
- returns the address of the first byte (return type is void, compatible with all data types)

```
char    *p1;
float   *p2;

p1 = malloc( 5 );
p2 = malloc( 4 );
```

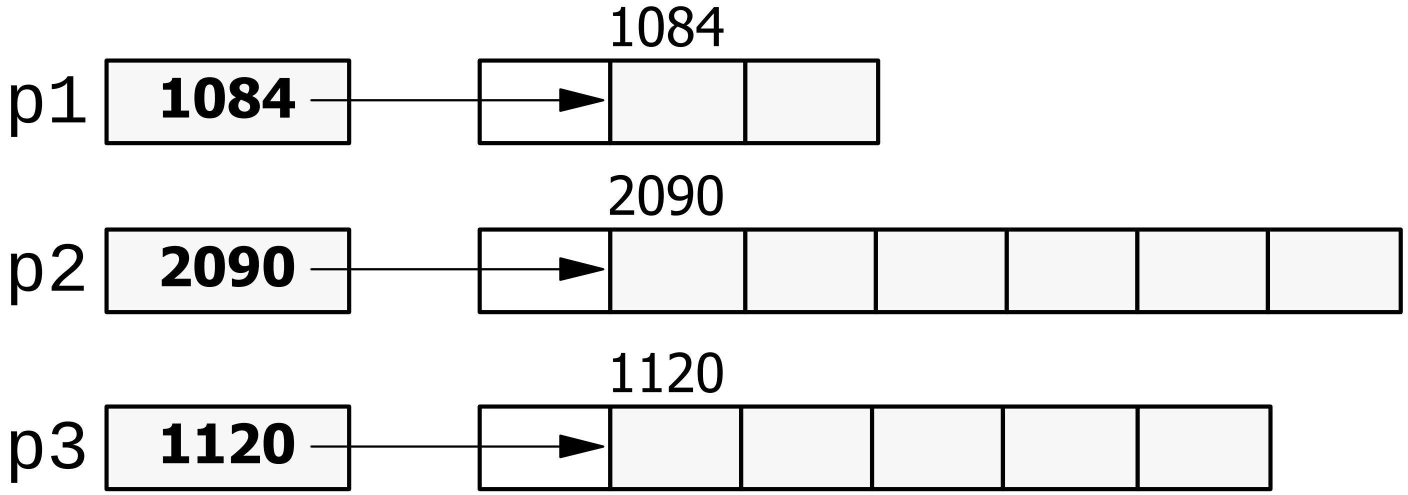
Dynamic Memory Allocation

```
int    *p1;

p1 = malloc( 2 );
*p1 = 12345;
```



Dynamic Memory Allocation

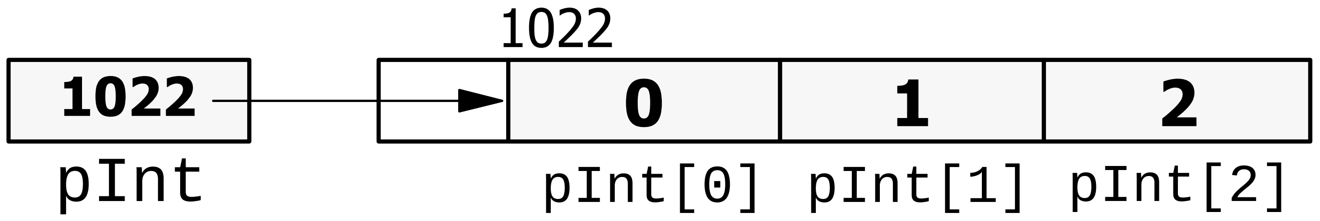
```
int        *p1, *p2;
char       *p3;

p1 = malloc( sizeof(int) );
p2 = malloc( 3*sizeof(int) );
p3 = malloc( 5*sizeof(char) );
```



S. Prasitjutrakul

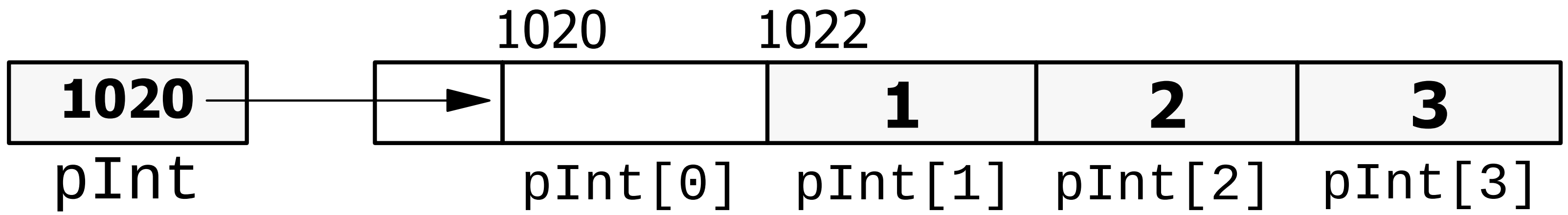
Dynamic Memory Allocation

```
int    *pInt;

pInt = malloc( 3*sizeof(int) );
for (i=0; i<3; i++) *(pInt+i) = i;
for (i=0; i<3; i++)   pInt[i] = i;
```



Dynamic Memory Allocation

```
int    *pInt;

pInt = malloc( 3*sizeof(int) );
pInt--;
for (i=1; i<=3; i++) *(pInt+i) = i;
for (i=1; i<=3; i++)   pInt[i] = i;
```

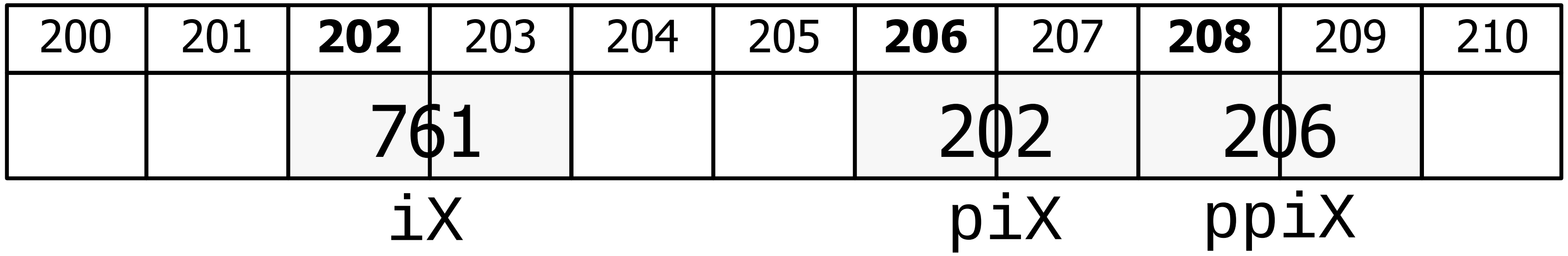


```
pInt[0] = 0; /* WRONG !! */
```

Returning Allocated Memory

```
int    *pInt, i;

for (i=0; i<=10000; i++) {
  pInt = malloc( 1000*sizeof(int) );
  ...
}

for (i=0; i<=10000; i++) {
  pInt = malloc( 1000*sizeof(int) );
  ...
  free( pInt );
}
```

Pointers to Pointers

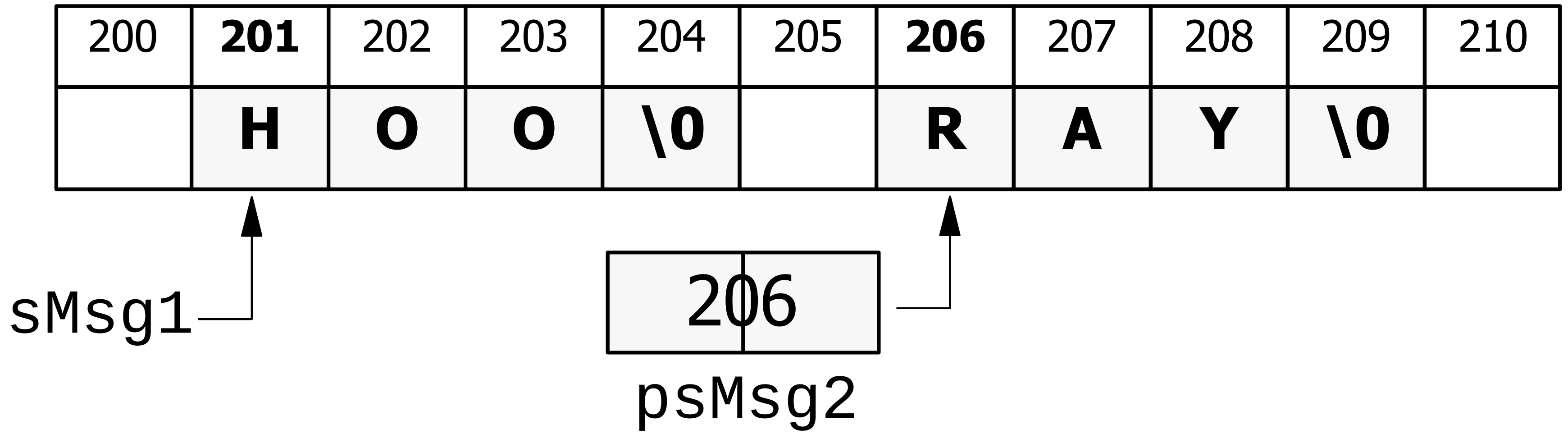


```
int    iX, *piX, **ppiX;

iX   = 761;
piX  = &iX;  *piX   = 761;
ppiX = &piX; **ppiX = 761;
```

Two levels of indirection

Strings and Pointers

```
char  sMsg1[] = "HOO";
char  *psMsg2 = "RAY";
```



Strings and Pointers

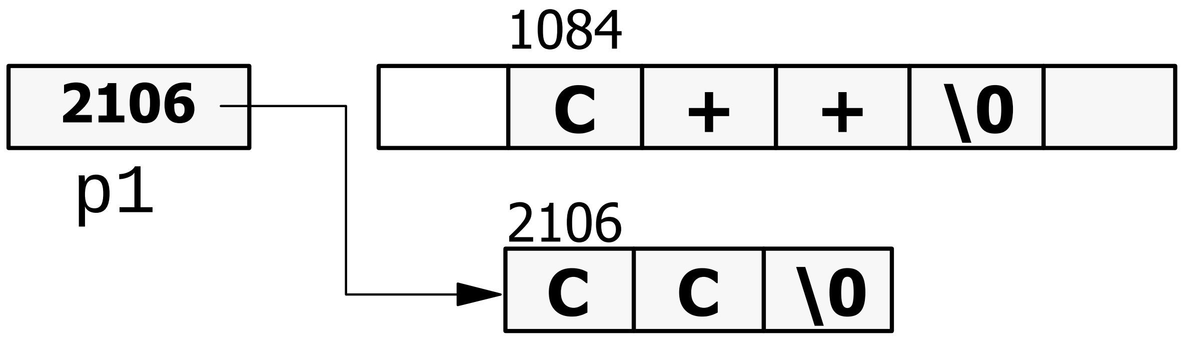
```
char  sMsg1[] = "HOO";
char  *psMsg2 = "RAY";

strcpy( psMsg2, "HEY" ); /* OK    */
strcpy( sMsg1, "HEY" );  /* OK    */
psMsg2 = "HELLO";        /* OK    */
sMsg1  = "HEY";          /* WRONG */

/* sMsg1  -- constant */
/* psMsg2 -- variable */
```

String == Pointer to char

S. Prasitjutrakul

Strings and Pointers

```
char    *p1;

p1 = malloc( 5*sizeof(char) );
strcpy(p1,"C++");
p1 = "CC";    /*  OK     */
```

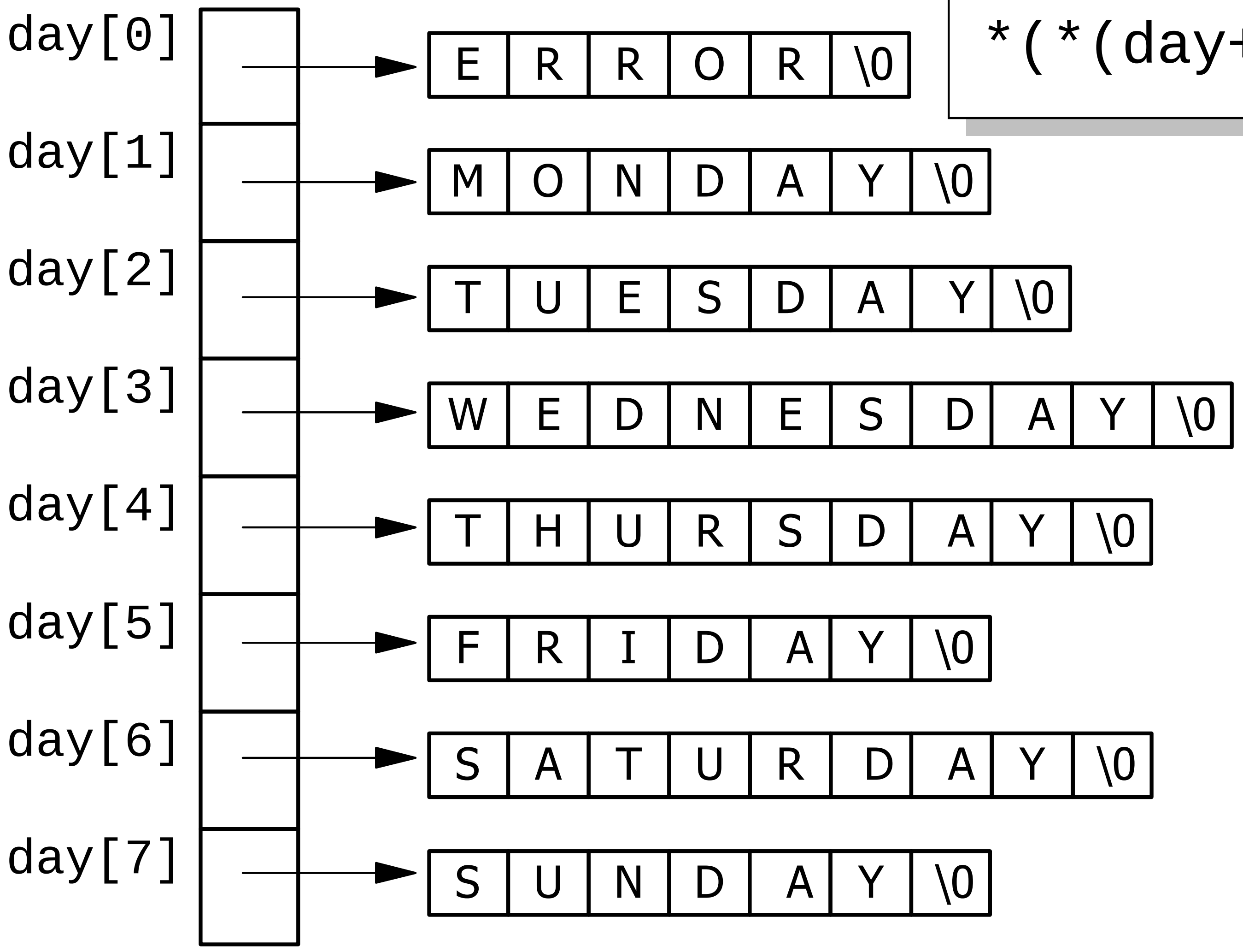


Array of Strings

```
char  *day[] = {"ERROR",
                 "MONDAY",
                 "TUESDAY",
                 "WEDNESDAY",
                 "THURSDAY",
                 "FRIDAY",
                 "SATURDAY",
                 "SUNDAY" };
```

Array of strings == Array of pointers to char

Array of Strings

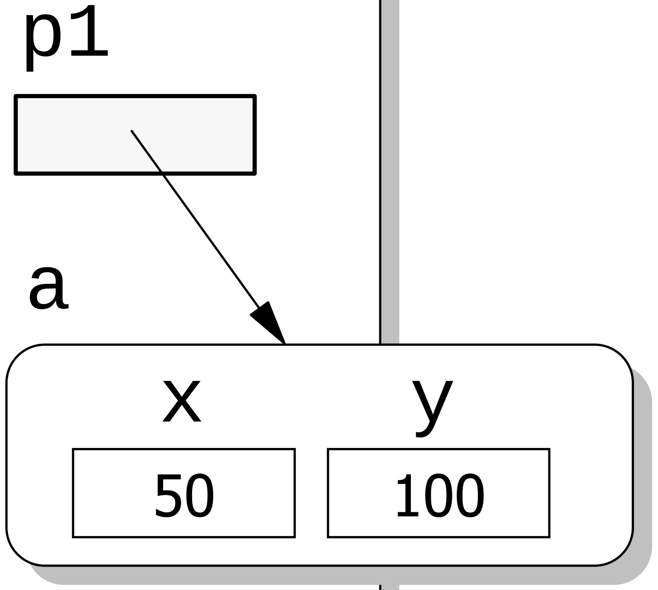


```
*(*(day+3)+2) = ?
```

S. Prasitjutrakul

Pointers to Structures

```
struct point {
    int   x;
    int   y;
};
...
struct point  *p1;
struct point  a;

a.x = 50;
a.y = 100;

p1 = &a;
(*p1).x = 50;
(*p1).y = 100;
```



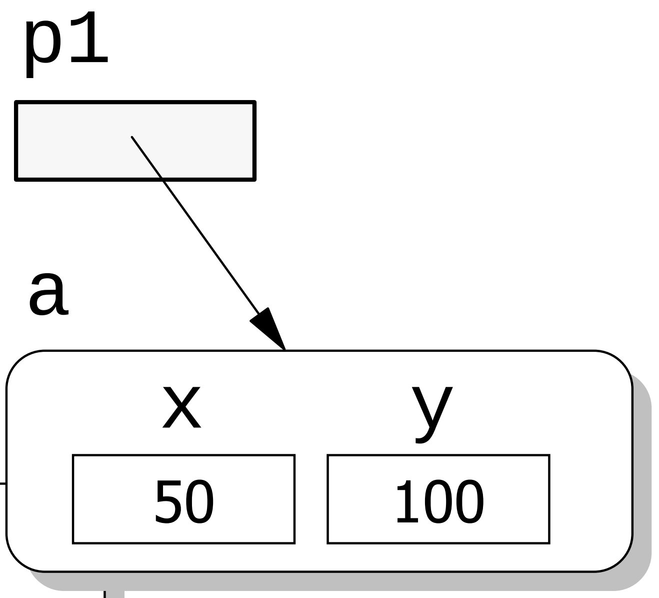
S. Prasitjutrakul

Pointers to Structures

```
a.x = 50;
a.y = 100;
```

```
p1 = &a;
(*p1).x = 50;
(*p1).y = 100;
```

```
p1 = &a;
p1->x = 50;
p1->y = 100;
```

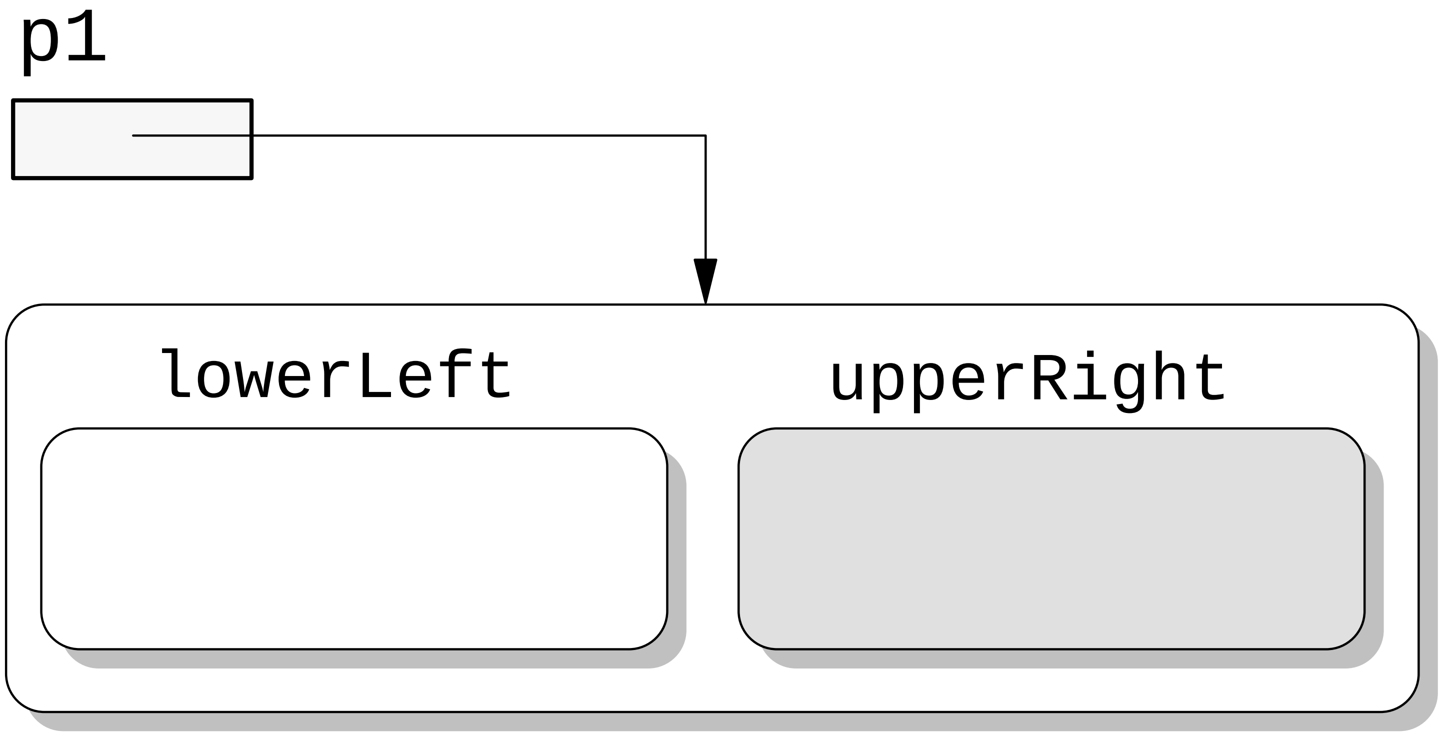


`pointer->structMember`

Pointers to Structures

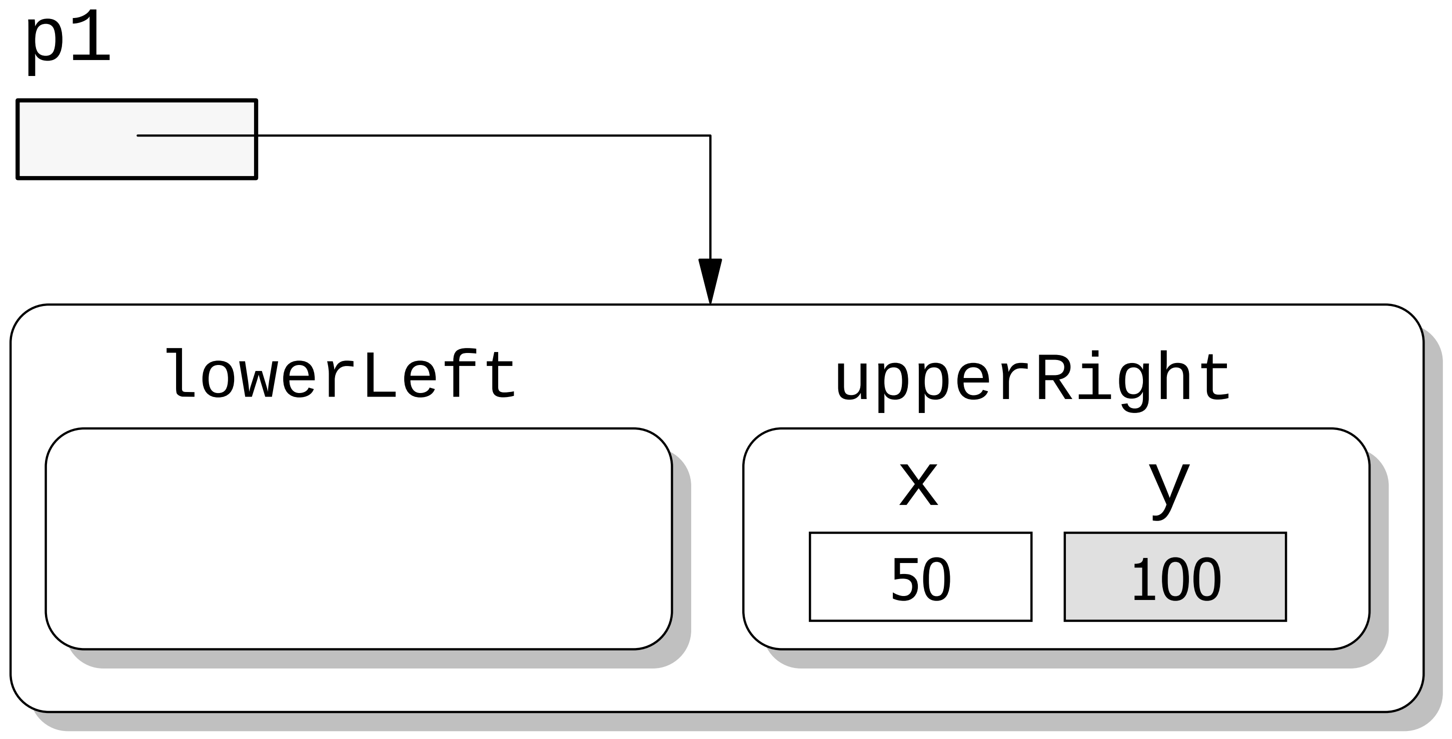
```
struct rect {
    struct point lowerLeft;
    struct point upperRight;
};
...
struct rect  box, *p1;
...
w = box.upperRight.x - box.lowerLeft.x;
h = box.upperRight.y - box.lowerLeft.y;

p1 = &box;
w = (p1->upperRight).x - (p1->lowerLeft).x;
h = (p1->upperRight).y - (p1->lowerLeft).y;
```

Pointers to Structures



```
p1->upperRight
```

Pointers to Structures



```
(p1->upperRight).y
```

S. Prasitjutrakul

Structures as Parameters

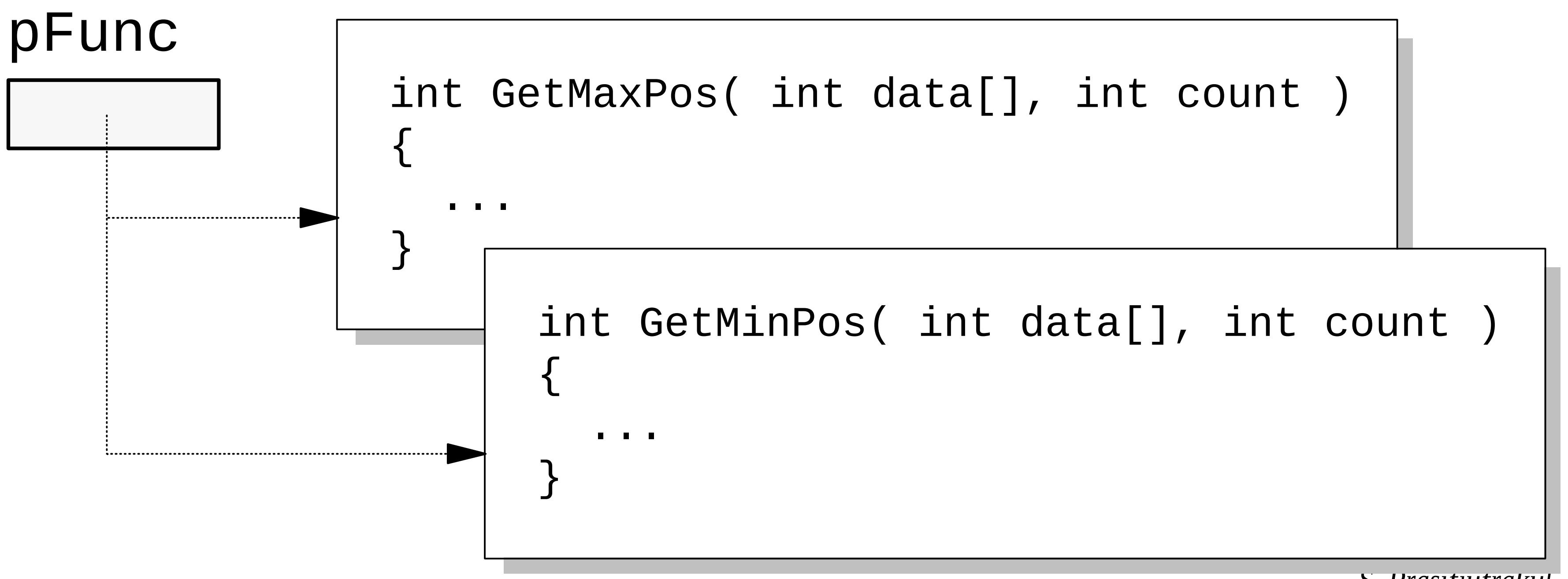
```
float Distance( struct point p1, struct point p2 )
{
  float     dx, dy;

  dx = p1.x - p2.x;
  dy = p1.y - p2.y;
  return ( sqrt( dx*dx + dy*dy ) );
}
```

```
float Distance( struct point *p1, struct point *p2 )
{
  float     dx, dy;

  dx = p1->x - p2->x;
  dy = p1->y - p2->y;
  return ( sqrt( dx*dx + dy*dy ) );
}
```

Faster

Pointers to Functions

- Hold the starting address of a function.
- Provide a flexible way of calling functions
- Allow a program to "pick" among several functions based on the circumstances.



```
int GetMaxPos( int data[], int count )
{
   ...
}
```

```
int GetMinPos( int data[], int count )
{
   ...
}
```

Pointers to Functions

```
int GetMaxPos( int data[], int count )
{
  ...
}
```

```
int GetMinPos( int data[], int count )
{
  ...
}
```

```
int  (*pFunc)( int data[], int count );

pFunc = GetMaxPos;
...
pFunc = GetMinPos;
...
x = pFunc( a, 7 );
```

Pointers to Functions

```
/* pFunc is a pointer to a function
   returning an integer.           */

int  (*pFunc)( int data[], int count );
```

```
/* pFunc is a function returning
   a pointer to an integer.        */

int  *pFunc( int data[], int count );
```

Selection Sort : Ascending Order

```
void SelectionSort( int data[], int count )
{
  if ( count > 1 ) {
     maxPos = GetMaxPos( data, count );
     SwapData( data, maxPos, count-1 );
     SelectionSort( data, count-1 );
  }
}

int GetMaxPos( int data[], int count )
{
  int     maxPos, i;
  maxPos = 0;
  for ( i=1; i<count; i++ )
    if ( data[maxPos] < data[i] ) maxPos = i;
  return maxPos;
}
```

Ascending order

Selection Sort : Descending Order

```
void SelectionSort( int data[], int count )
{
  if ( count > 1 ) {
     minPos = GetMinPos( data, count );
     SwapData( data, minPos, count-1 );
     SelectionSort( data, count-1 );
  }
}

int GetMinPos( int data[], int count )
{
  int     minPos, i;
  minPos = 0;
  for ( i=1; i<count; i++ )
    if ( data[minPos] > data[i] ) minPos = i;
  return minPos;
}
```

Descending order

Pointers to Functions

```
int GetMinPos( int data[], int count );

void SelectionSort( int data[], int count )
{
  int (*pFunc)( int data[], int count );

  pFunc = GetMinPos;
  if ( count > 1 ) {
     minPos = pFunc( data, count );
     SwapData( data, minPos, count-1 );
     SelectionSort( data, count-1 );
  }
}
```

Pointers to Functions

```
/*      function prototypes      */
int GetMinPos( int data[], int count );
int GetMaxPos( int data[], int count );

void SelectionSort( int data[], int count,
                    int (*pF)( int data[], int count ) )
{
  if ( count > 1 ) {
     mPos = pF( data, count );
     SwapData( data, mPos, count-1 );
     SelectionSort( data, count-1, pF );
  }
}
```

```
SelectionSort( list1, 100, GetMinPos );
...
SelectionSort( list2, 510, GetMaxPos );
```