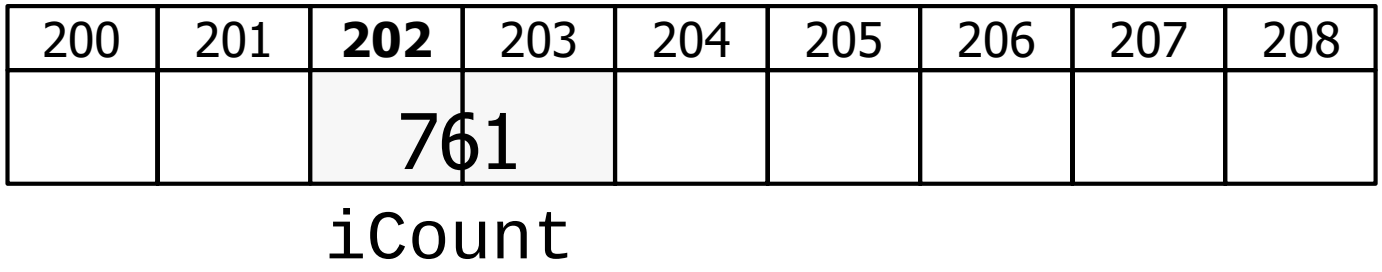
Outline

- Computer memory addresses
- Pointers and addresses
- Pointers and function arguments
- Pointers and arrays
- Pointer arithmetic
- Dynamic memory allocation
- Pointers to pointers
- Pointers to structures
- Pointers to functions

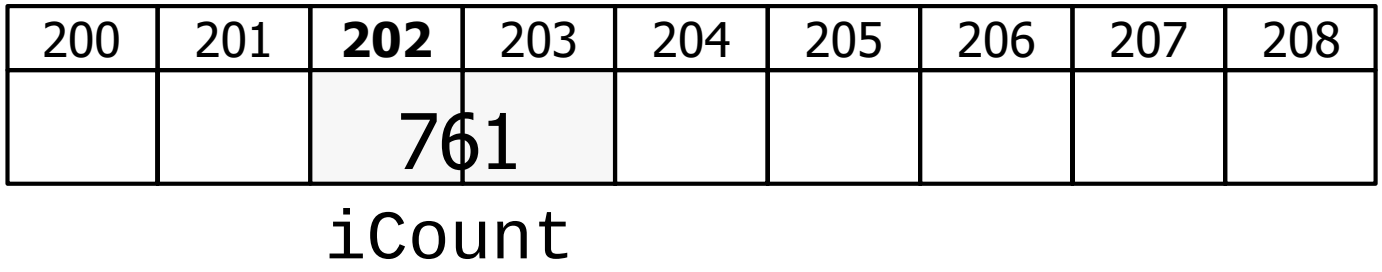
Computer Memory : Address




```
int    iCount;

iCount = 761;
printf("data : %d\n",  iCount );
printf("addr : %d\n", &iCount );
```

" address of "

Pointer

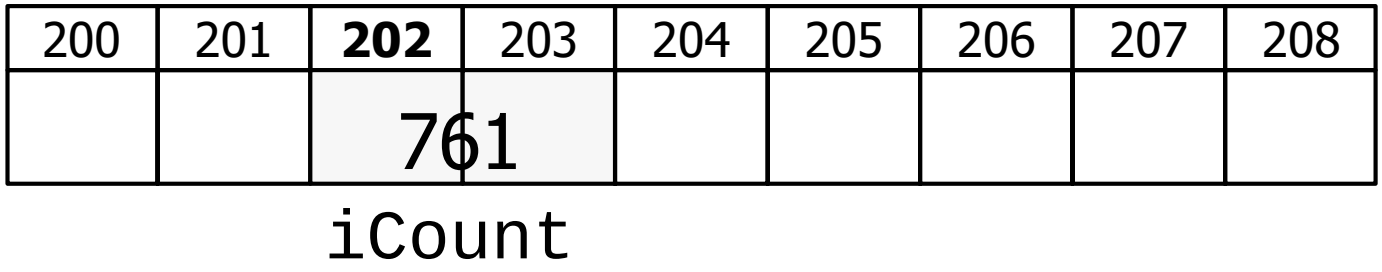



```
int    iCount;
int    *pC;

pC = &iCount;     /* pC = 202 */
```

pC is a variable containing the address of an integer.

pC is said to "point to" iCount.

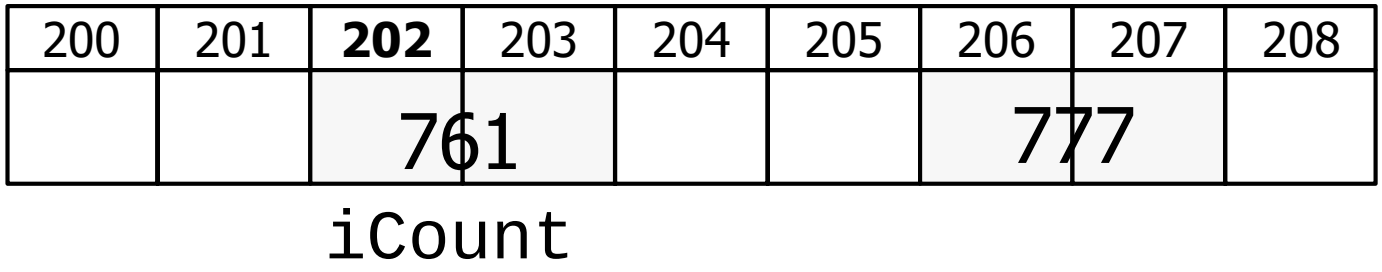
Pointer




```
int    iCount;
int    *pC;

pC  = &iCount;     /* pC = 202 */
*pC = 761;
```

Assign 761 to wherever the pC points.

Pointer




```
int    iCount;
int    *pC;

pC  = 206;     /* Don't do this */
*pC = 777;
```

Passing by Value

```
#include   <stdio.h>

void Set2Zero( int a )
{
  a = 0;
}

main()
{
  int      x;

  x = 9;
  Set2Zero( x );
  printf("x = %d\n", x );
}
```

Passing by Reference

```
#include   <stdio.h>

void Set2Zero( int *pa )
{
  *pa = 0;
}

main()
{
  int      x;

  x = 9;
  Set2Zero( &x );
  printf("x = %d\n", x );
}
```

Pointers

```
int       x, y;
int       *pX, *pY;

x = y = 0;

pX  = &x;  *pX = 7;   /* x = 7,  y = 0 */

pY  = &y;  *pY = 8;   /* x = 7,  y = 8 */

*pX = *pY;            /* x = 8,  y = 8 */

pX  = pY;  *pX = 6;   /* x = 8,  y = 6 */
```

Example : swap()

```
void swap( int a, int b )
{
  int   t;

  t = a;
  a = b;
  b = t;
}
```

```
void swap( int *pa, int *pb )
{
  int   t;

  t   = *pa;
  *pa = *pb;
  *pb = t;
}
```

Example : swap()

```
void swap( int *pa, int *pb )
{
  int   t;

  t   = *pa;
  *pa = *pb;
  *pb = t;
}
```

```
void swap( int *pa, int *pb )
{
  int   *pt;

  *pt = *pa;     /* WRONG  */
  *pa = *pb;
  *pb = *pt;
}
```

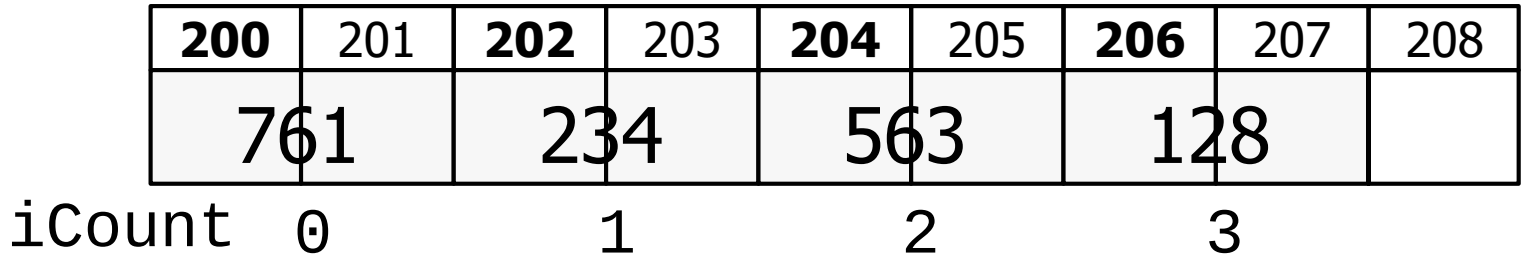
Pointers and Variable Types

```
int     iCount;
float   fRate;
char    cSelection;

int     *pC;
float   *pR;
char    *pS;

pC = &iCount;
pR = &fRate;
pS = &cSelection;
```

Size of Pointers

```
 ...
int     *pC;
float   *pR;
char    *pS;
 ...

printf( "size of (int *) = %d\n",   sizeof(int *) );
printf( "size of (float *) = %d\n", sizeof(float *) );
printf( "size of (char *) = %d\n",  sizeof(char *) );
 ...
```

Addresses of Array Elements

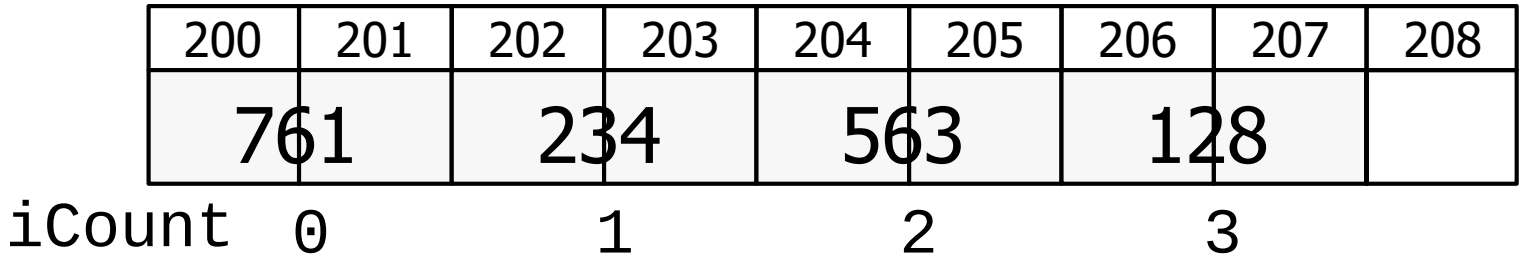



```
int    iCount[4];

printf( "&iCount[0] = %d\n", &iCount[0] );
printf( "&iCount[1] = %d\n", &iCount[1] );
printf( "&iCount[2] = %d\n", &iCount[2] );
printf( "&iCount[3] = %d\n", &iCount[3] );
```

Addresses of Array Elements

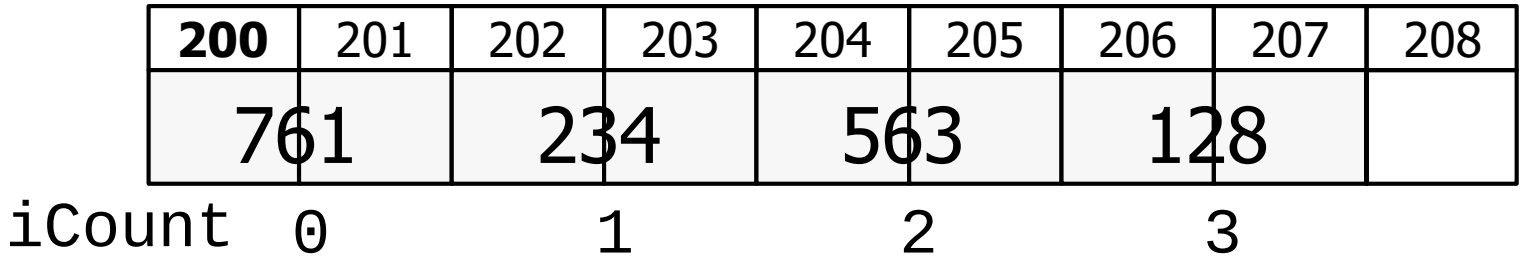
```
int      i;
char     cData[10];
int      iData[10];
float    fData[10];
double   dData[10];

printf( n\tchar\tint\tfloat\tdouble\n");
for (i=0; i<10; i++)
  printf( "%d\t%d\t%d\t%d\t%d\n",
          i, &cData[i], &iData[i],
             &fData[i], &dData[i] );
```

Pointers and Arrays



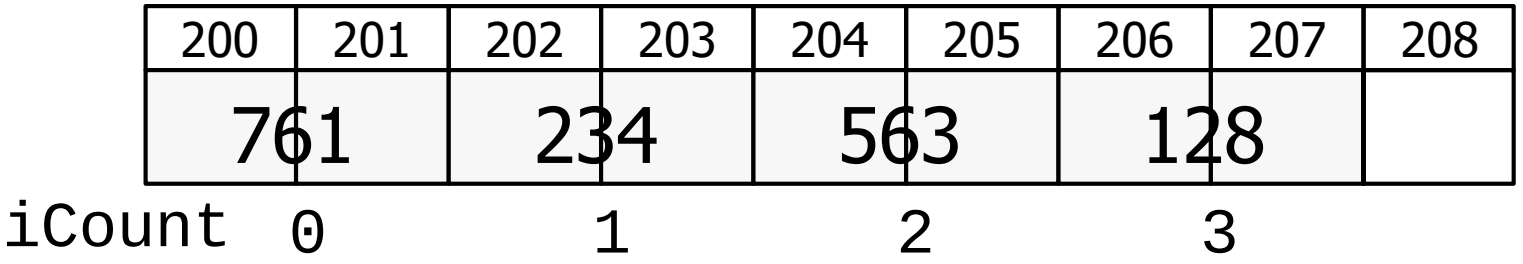

```
int    iCount[4];
int    *pC;

pC = &(iCount[0]);  *pC = 761;
pC = &(iCount[1]);  *pC = 234;
pC = &(iCount[2]);  *pC = 563;
pC = &(iCount[3]);  *pC = 128;
```

Pointers and Arrays



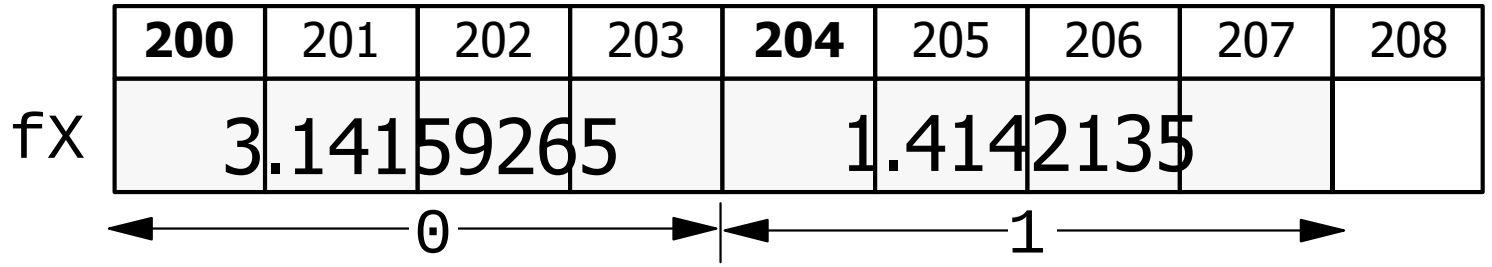

```
int    iCount[4];
int    *pC1, pC2

pC1 = &(iCount[0]);
pC2 = iCount;
printf( "%d  %d\n", pC1, pC2 );
```

Pointers and Arrays



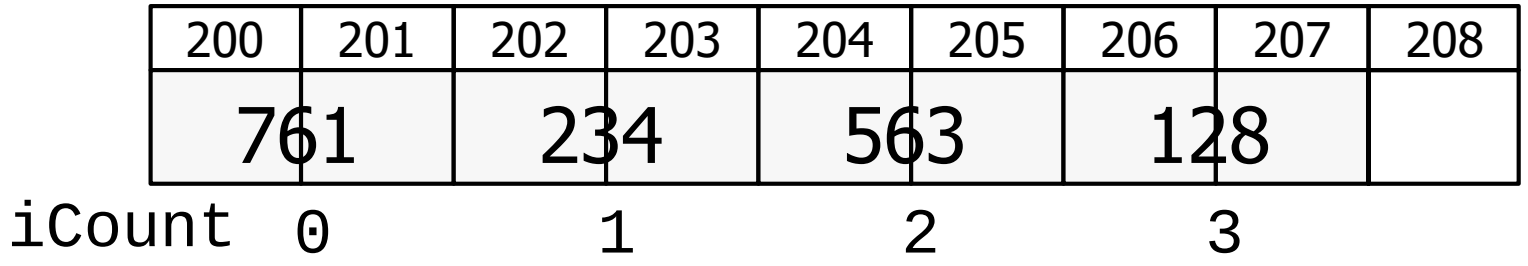

```
int    iCount[4];
int    *pC;

pC = iCount;          /* pC = 200 */
pC++;  *pC = 234;     /* pC = 202 */
pC++;  *pC = 563;     /* pC = 204 */
*(++pC) = 128;        /* pC = 206 */
```

Pointers and Arrays



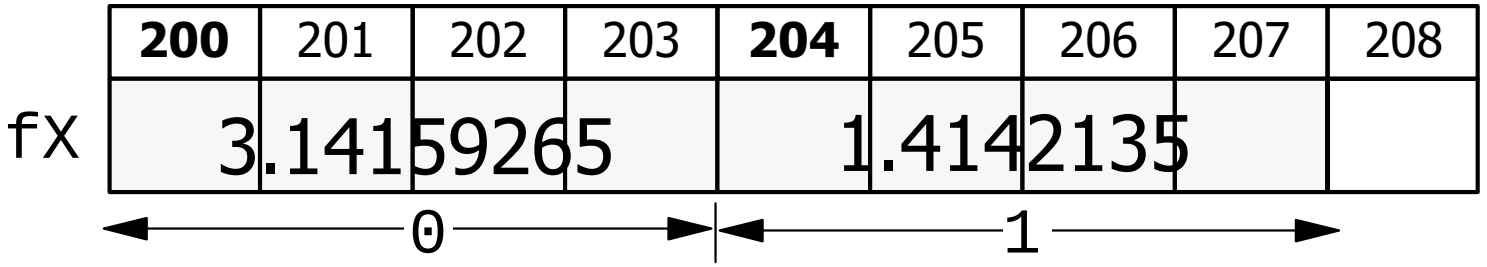

```
float    fX[2];
float    *pC;

pC = fX;          /* pC = 200 */
pC++;             /* pC = 204 */
*pC = 1.4142135;
```

Pointers and Arrays



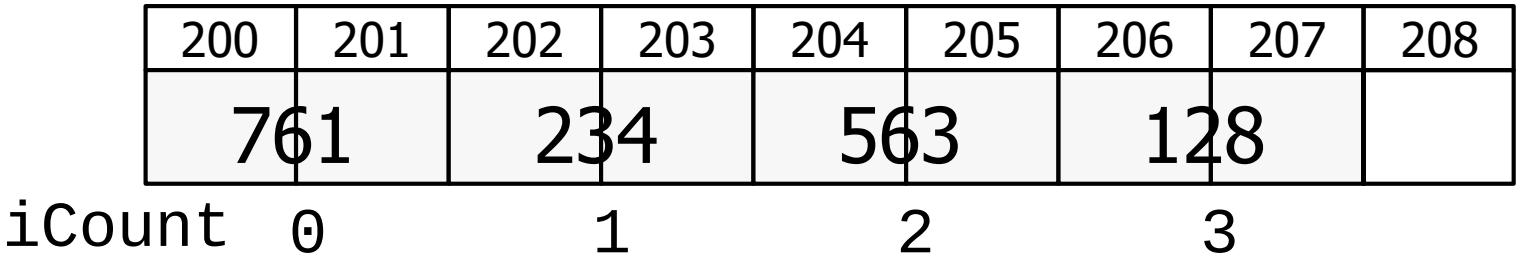

```
int    iCount[4];
int    *pC;

*(iCount+1) = 234;  /* iCount[1] == *(iCount+1) */
*(iCount+2) = 563;  /* iCount[2] == *(iCount+2) */

pC = iCount;    *(pC+2) = 563;  /* iCount[2] */

pC = iCount+1;  *(pC+2) = 128;  /* iCount[3] */
```

Pointers and Arrays




```
float    fX[2];
float    *pC;

pC = &fX[1];          /* pC = 204 */
pC--;                 /* pC = 200 */
*pC = 3.14159265;

pC = &fX[1];
*(pC-1) = 3.14159265;
```

Differencing




```
int    iCount[4];
int    *p1, *p2;
int    diff;

p1 = &iCount[3];    /* p1 = 206 */
p2 = &iCount[0];    /* p2 = 200 */
diff = p1 - p2;     /* diff = 3 */
```

Comparison

- If `p1` and `p2` point to elements of the same array, the comparison

```
p1 < p2
```

  is true if `p1` points to the an earlier member of the array than `p2` does.

Do & Don't

- Don't use an uninitialized pointer.
- Don't try to do mathematical operations other than `+, ++, -, --`.
- Don't try to increment or decrement an array variable

```
int   a[10];
a++;
```

- Do understand the size of variable types.

Passing Arrays to Functions

```
int largest( int x[], int count )
{
  int    i, maxVal;

  maxVal = x[0];
  for (i=1; i<count; i++)
    if ( x[i] > maxVal ) maxVal = x[i];
  return  maxVal;
}
void main( void )
{
  int   a[100];
  ...
  maxA1 = largest( a, 50 );
  maxA2 = largest( &(a[10]), 50 );
```

Passing Arrays to Functions

```
int largest( int *px, int count )
{
  int    i, maxVal;

  maxVal = *px;
  for (i=1; i<count; i++)
    if ( *(px+i) > maxVal ) maxVal = *(px+i);
  return  maxVal;
}
void main( void )
{
  int   a[100];
  ...
  maxA1 = largest( a, 50 );
  maxA2 = largest( &(a[10]), 50 );
```