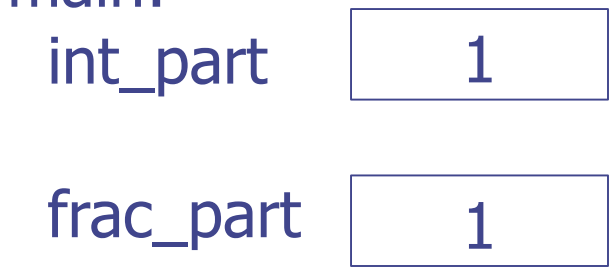
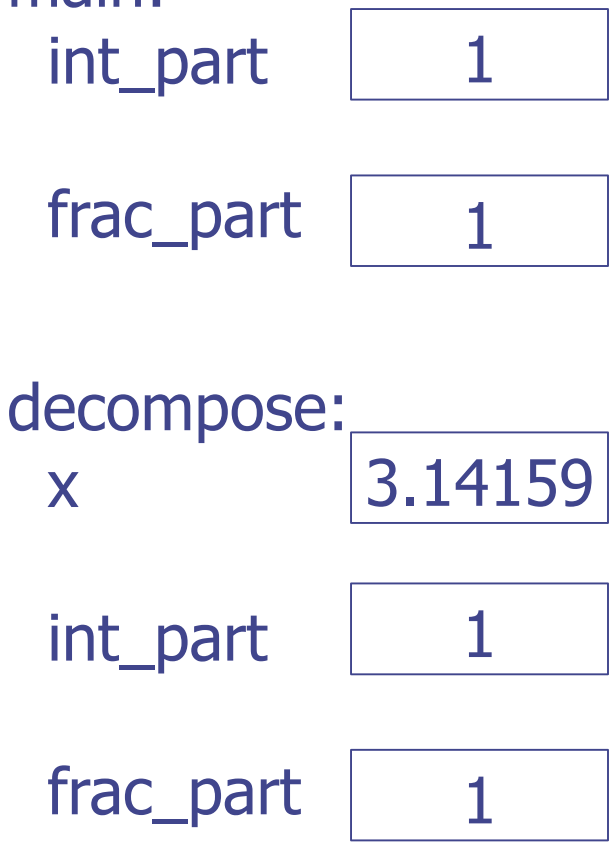
Lesson 6: Pointer

Jiun-Long Huang
Department of Computer Science
National Chiao Tung University

```
#include <stdio.h>

void decompose(double x, long int_part,
               double frac_part)
{
    int_part = (long) x;
    frac_part = x - int_part;
}

int main(void)
{
  long int_part=1;
  double frac_part=1;
  decompose(3.14159,int_part,frac_part);
  printf("%ld %lf\n",int_part,frac_part);
  return 0;
}
```

```
1 1.000000
```


main:
int_part
1
frac_part
1


main:
int_part
1
frac_part
1
decompose:
x
3.14159
int_part
1
frac_part
1

main:

int_part	1
frac_part	1

decompose:

x	3.14159
int_part	3
frac_part	0.14159

main:

int_part 1

frac_part 1

decompose:

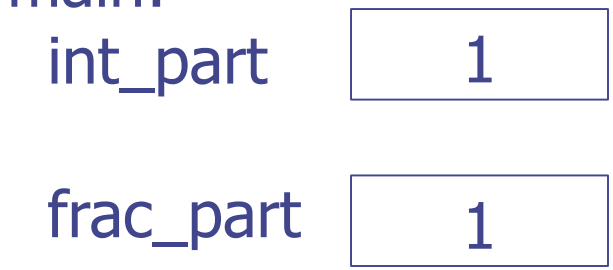
x

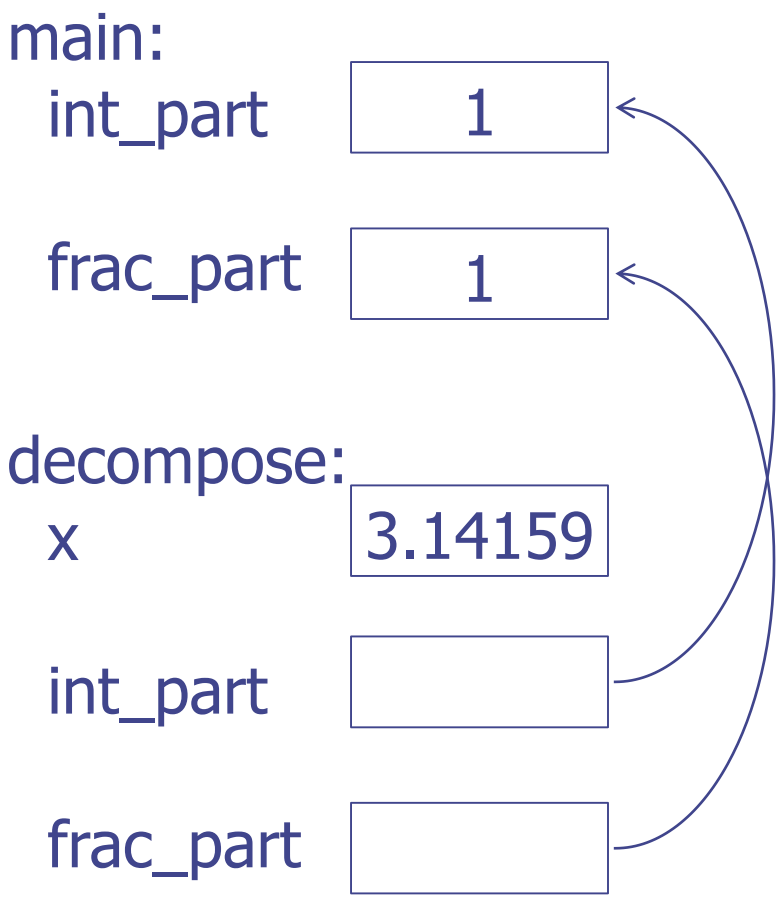
int_part

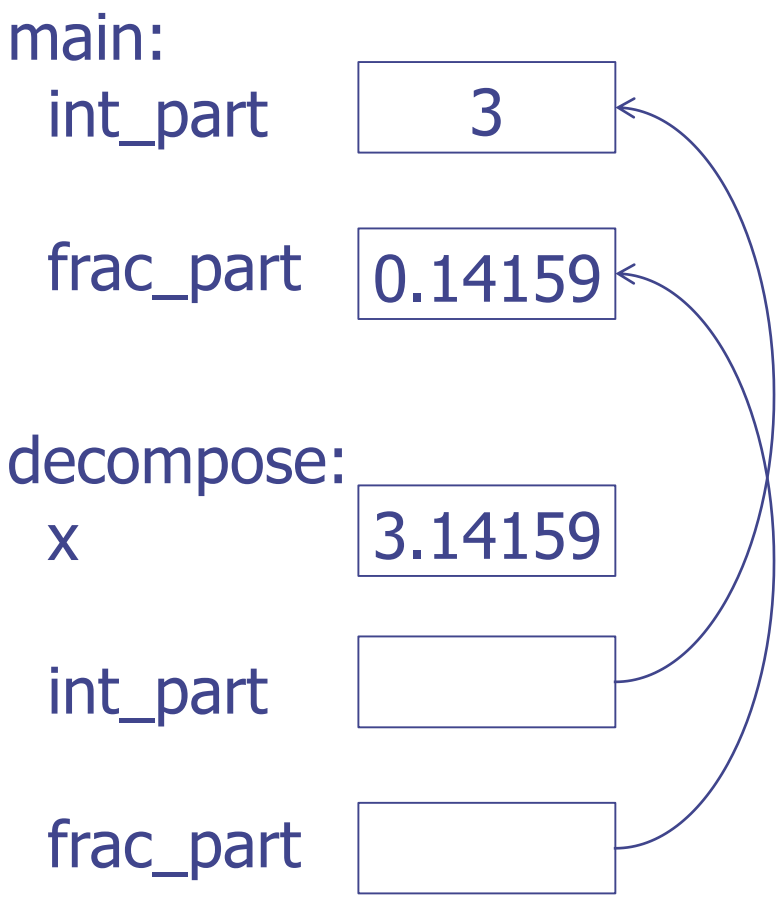
frac_part

```c
#include <stdio.h>

void decompose(double x, long *int_part,
               double *frac_part)
{
    *int_part = (long) x;
    *frac_part = x - *int_part;
}

int main(void)
{
  long int_part=1;
  double frac_part=1;
  decompose(3.14159,&int_part,&frac_part);
  printf("%ld %lf\n",int_part,frac_part);
  return 0;
}
```

```
3 0.141590
```


main:
int_part
1
frac_part
1


main:
int_part
1
frac_part
1
decompose:
x
3.14159
int_part
frac_part


main:
int_part
3
frac_part
0.14159
decompose:
x
3.14159
int_part
frac_part

```c
#include <stdio.h>

void decompose(double x, long int_part[],
               double frac_part[])
{
    int_part[0] = (long) x;
    frac_part[0] = x - int_part[0];
}

int main(void)
{
  long int_part[]={1};
  double frac_part[]={1};
  decompose(3.14159,int_part,frac_part);
  printf("%ld %lf\n",int_part[0],frac_part[0]);
  return 0;
}
```

```
3 0.141590
```

Little Endian

- "Little Endian" means that the low-order byte of the number is stored in memory at the lowest address, and the high-order byte at the highest address. (The little end comes first.)

"An Essay on Endian Order," http://www.cs.umass.edu/~verts/cs32/endian.html

- A 4 byte LongInt Byte3 Byte2 Byte1 Byte0 will be arranged in memory as follows:
  - Base Address+0 Byte0
  - Base Address+1 Byte1
  - Base Address+2 Byte2
  - Base Address+3 Byte3
- Intel processors (those used in PC's) use "Little Endian" byte order.

Big Endian

- "Big Endian" means that the high-order byte of the number is stored in memory at the lowest address, and the low-order byte at the highest address. (The big end comes first.)

- LongInt, would then be stored as:
  - Base Address+0 Byte3
  - Base Address+1 Byte2
  - Base Address+2 Byte1
  - Base Address+3 Byte0
- Motorola processors (those used in Mac's) use "Big Endian" byte order.

Little Endian and Big Endian

- $(10000)_{10}=(0010011100010000)_2=(2710)_{16}$
- Little Endian

int a=10000;

00010000	00100111	00000000	00000000

- Big Endian

int a=10000;

00000000	00000000	00100111	00010000

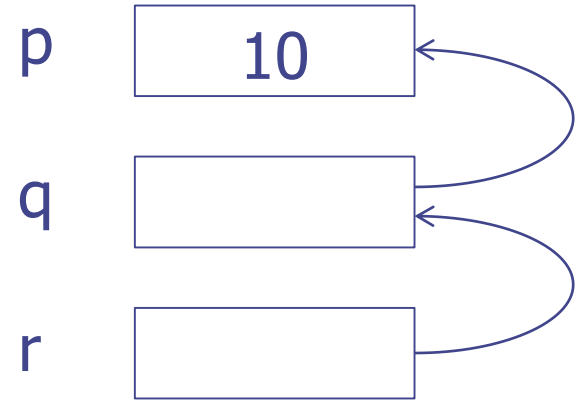
```
#include <stdio.h>
int a=10000;
int main()
{
  char * p=(char *)&a;
  printf("%x %x %x %x",*p, *(p+1), *(p+2),*(p+3));
  return 0;
}
```

```
10 27 0 0
```

Common File Formats and Their Endian Order

- Adobe Photoshop -- Big Endian
- BMP (Windows and OS/2 Bitmaps) -- Little Endian
- GIF -- Little Endian
- JPEG -- Big Endian
- MacPaint -- Big Endian
- PCX (PC Paintbrush) -- Little Endian
- Microsoft RIFF (.WAV & .AVI) -- Both
- TGA (Targa) -- Little Endian
- TIFF -- Both, Endian identifier encoded into file
- …

```c
#include <stdio.h>

int main(void)
{
  int p=10, *q, **r;
  q=&p;
  r=&q;
  printf("%d %d %d\n", p, *q, **r);
  return 0;
}
```



Constant Pointer

- https://cdecl.org/
  - C gibberish ↔ English
- const int x
  - declare x as const int
- int const x
  - declare x as const int

- const int * x
  - declare x as pointer to const int
- int const * x
  - declare x as pointer to const int
- int * const x
  - declare x as const pointer to int
- const int * const x
  - declare x as const pointer to const int

- Constant variables can be initialized only in declaration.

```
int main(void)
{
  const int n;
  n=5;  // Compilation error
  return 0;
}
```

```
int main(void)
{
  const int n=5;
  return 0;
}
```

```
int main(void)
{
  const int m=5;
  const int n=6;
  const int *p;
  p=&m;
  *p+=1;  // Compilation error
  p=&n;   // OK
  return 0;
}
```

```
int main(void)
{
  const int m=5;
  const int n=6;
  int * const p=(int *)&m;
  *p+=1;  // OK
  p=&n;   // Compilation error
  return 0;
}
```