CS 33

Introduction to C
Part 6
Numeric Conversions

```c
short a;
int b;
float c;

b = a;    /* always works */
a = b;    /* sometimes works */
c = b;    /* sort of works */
b = c;    /* sometimes works */
```
Implicit Conversions (1)

```c
float x, y=2.0;
int i=1, j=2;

x = i/j + y;
/* what's the value of x? */
```
Implicit Conversions (2)

```c
float x, y=2.0;
int i=1, j=2;
float a, b;

a = i;
b = j;
x = a/b + y;
/* now what's the value of x? */
```
Explicit Conversions: Casts

```c
float x, y=2.0;
int i=1, j=2;

x = (float)i/(float)j + y;
/* and now what's the value of x? */
```
Fun with Functions (1)

```c
void ArrayDouble(int A[], int len) {
    int i;
    for (i=0; i<len; i++)
        A[i] = 2*A[i];
}
```
Fun with Functions (2)

```c
void ArrayBop(int A[],
    int len,
    int (*func)(int)) {
    int i;
    for (i=0; i<len; i++)
}
```
Fun with Functions (3)

```c
int triple(int arg) {
   return 3*arg;
}

int main() {
   int A[20];
   ... /* initialize A */
   ArrayBop(A, 20, triple);
   return 0;
}
```
Swap, Revisited

```c
void swap(int *i, int *j) {
    int *tmp;
    tmp = j; j = i; i = tmp;
}
/* can we make this generic? */
```
Casts, Revisited

• Two purposes
  – coercion
    
    ```
    int i, j;
    float a; /* sizeof(float) == 4 */
    a = (float)i/(float)j;
    ```
  – intimidation
    
    ```
    float x, y;
    swap((int *)&x, (int *)&y);
    ```

  done for primitive types
  done for pointer types
Quiz 1

• Will this work?

```c
double x, y; // sizeof(double) == 8
...
swap((int*)&x, (int*)&y);
```

a) yes
b) no
Intimidation
Nothing, and More …

• *void* means, literally, nothing:

```c
void NotMuch(void) {
    printf("I return nothing\n");
}
```

• What does *void* * mean?
  – it’s a pointer to anything you feel like
    » a generic pointer
Rules

• Use with other pointers
  
  ```c
  int *x;
  void *y;
  x = y; /* legal */
  y = x; /* legal */
  ```

• Dereferencing
  
  ```c
  void *z;
  *z; /* illegal! */
  ```
An Application: Generic Swap

```c
void gswap (void *p1, void *p2, int size) {
    int i;
    for (i=0; i < size; i++) {
        char tmp;
        tmp = (*((char *)p1))[i];
        (*((char *)p1))[i] = (*((char *)p2))[i];
        (*((char *)p2))[i] = tmp;
    }
}
```
Using Generic Swap

```c
short a, b;
gswap(&a, &b, sizeof(short));

int x, y;
gswap(&x, &y, sizeof(int));

int A[] = {1, 2, 3}, B[] = {7, 8, 9};
gswap(A, B, sizeof(A));
```
For Our Next Trick ...

What’s my type?

```c
int *f0(int *a) {
    ...
}

float *f1(float *a) {
    ...
}

char *f2(char *a) {
    ...
}
```
Working Our Way There …

• An array of 3 ints
  – int A[3];

• An array of 3 int *s
  – int *A[3];

• A func returning an int *, taking an int *
  – int *f(int *);

• A pointer to such a func
  – int *(pf)(int *);
There …

• An array of func pointers
  - `int *(*pf[3])(int *)`;

• An array of generic func pointers
  - `void *(*pf[3])(void *)`;
Using It

```c
int *f0(int *a) { *a += 1; return a; }
float *f1(float *a) { *a += 1; return a; }
char *f2(char *a) { *a += 1; return a; }
int main() {
    int x = 1;
    int *p;
    void *(*pf[3])(void *);
    pf[0] = (void *(*)(void *))f0;
    pf[1] = (void *(*)(void *))f1;
    pf[2] = (void *(*)(void *))f2;
    p = pf[0](&x);
    printf("%d\n", *p);
    return 0;
}
```

$ ./funcptr
2
$
int *f0(int *a) { *a += 1; return a; }
float *f1(float *a) { *a += 1; return a; }
char *f2(char *a) { *a += 1; return a; }
int main() {
    int x = 1;
    int *p;
    void *(*pf[3])(void *);
    pf[0] = (void *(*)(void *))f0;
    pf[1] = (void *(*)(void *))f1;
    pf[2] = (void *(*)(void *))f2;
    p = pf[1](&x); // was pf[0]
    printf("%d\n", *p);
    return 0;
}

What is printed?

a) 2
b) 2.5
c) something different from the above
d) nothing: syntax error
Casts, Yet Again

• They tell the C compiler: “Shut up, I know what I’m doing!”

• Sometimes true
  \[
  \text{pf}[0] = (\text{void} (*) (\text{void} *))f0;
  \]

• Sometimes false
  \[
  \text{long } f = 7;
  (\text{void}(*)(\text{int}))f(2);
  \]
Laziness ...

• Why type the declaration
  \[
  \text{void } (*f)(\text{void } *, \text{ void } *);
  \]
• You could, instead, type
  \[
  \text{MyType } f;
  \]
• (If, of course, you can somehow define
  \textit{MyType} to mean the right thing)
**typedef**

- Allows one to create new names for existing types

```c
typedef int *IntP_t;
```

```c
IntP_t x;
```

- means the same as

```c
int *x;
```
More typedefs

typedef struct complex {
  float real;
  float imag;
} complex_t;

complex_t i, *ip;
And ...

typedef void *(*MyFunc_t)(void *, void *)

MyFunc_t f;

// you must do its definition the long way

void *f(void *a1, void *a2) {
    ...
}