Pointers to functions

R. Inkulu
http://www.iitg.ac.in/rinkulu/

Function name and its address

```
int func(float f) { return f-2; }
int main(void) {
   printf("%d, %d, %d\n", func(5.23), (&func)(5.23),
          (*func)(5.23));
   //prints 3, 3, 3

   printf(%d, %d\n", sizeof(void*), sizeof(&func));
   //prints 4, 4

   printf("%p, %p\n", func, &func);
   //prints 0x804841d, 0x804841d
}
```

- address of function is the address from where the function definition is residing in the program text area of process memory
- convention: to get the address of a function, use & before the name of the function; to invoke a function using the address saved in a variable, dereference the address and pass-in the actual parameters

Function pointer variables

```
int handleEvent1(float) { ... }
void *func4(void* ptr, int i) { ... }
//custom types: pointers to functions
typedef int (*FuncTypeA)(float);
typedef void *(*FuncTypeB)(void*, int);

int main(void) {  int j, k; char *p;
   printf("%d, %d, %d\n",
   sizeof(void*), sizeof(FuncTypeA),
   sizeof(FuncTypeB)); //prints 4, 4, 4
   ...
   FuncTypeA abc = &handleEvent1;
   k = (*abc)(13.5);
   ...
   FuncTypeB funcB = &func4;
   p = (char*) (*funcB)(p, j); }
```

- *function pointer* (a.k.a. *pointer to a function*): a variable that stores the address of a function

Motivation: template function

```
typedef double (*FuncType)(double);
void computeValue(FuncType func, double p[], int
numelem) {
   double sum=0;
   for (int i=0; i<numelem; i++)
       sum += (*func)(p[i]);
   return sum; }

int main(int argc, char *argv[]) {
   double a[100];
   ...   //array a is initialized
   ...
   if (argc > 1)
      computeValue(&cos, a, 100);
   else
      computeValue(&tan, a, 100);
}
```

Motivation: callback functions

```
int funcA(double) { ... }
int funcB(double) { ... }
typedef int (*FuncType)(double);
int initFuncTable(FuncType p[][2]) {
   p[0][0] = &funcA; p[0][1] = &funcB;
   p[1][0] = &funcA; p[1][1] = &funcB;
}
int main(void) {
    int i, j, k;
    FuncType buf[2][2];
    initFuncTable(buf);
    ...   //computed values of i and j
          //define the event of interest
    k =  (*buf[i][j])(35.65);
    ...  }
```

- appropriate function is chosen in runtime based on the values of i and j

Array of pointers to varied sized arrays of function pointers

```
typedef void *(*FuncType)(void);

void *func(void) { ... }

void funcA(int count) {
   FuncType *buf[2];
   for (int i=0; i<2; i++)
      buf[i] = (FuncType*)
         malloc((count+i)*sizeof(FuncType));
   ...
   *(buf[1]+3) = &func;  //assuming count>=4
   ...
   char *ptr = (char*) (*buf[1][3])();
   ...
   for (int i=0; i<2; i++)
      free(buf[i]);
}
```

Array of pointers to fixed sized arrays of function pointers

```
 typedef void *(*FuncType)(double);

 void *func(double) { ... }

 void funcA(int count){
    FuncType (*buf)[2];

    buf = (FuncType (*)[2])
          malloc(count*sizeof(FuncType[2]));
    ...
    buf[1][1] = &func;  //assuming count>=2
    ...
    char *ptr = (char*) (*buf[1][1])(789.80);
    ...
    free(buf);
}
```

More declarations

```c
typedef char (*(*A)(void))[5];
//type A is a function pointer that takes in
//void parameters and returns a pointer to char [5]

char (*f1(void))[5] { ... }
char (*f2(void))[5] { ... }
char (*f3(void))[5] { ... }

int main(void) {
   A b[3];  char (*p)[5];
   b[0] = &f1; b[1] = &f2; b[2] = &f3;
   ...
   p = (*b[1])();
   ...
}
```

More declarations (cont)

homework: function accepts double and returns pointer to array[] of pointers each of which points to a function that returns char and accepts float