C Pointers

Ned Nedialkov

McMaster University
Canada

SE 3F03
March 2013

Outline

Pointers to pointers

Pointers to functions

Arrays of pointers

More examples

Pointers to pointers

```
int i;
int *pi;    // pointer to int
int **ppi; // pointer to pointer to int
pi = &i;    // pi points to i
int a[10];
//...
pi = a;
int b = *(pi+2);
```

Pointers to functions

```
#include<stdio.h>
int sqr(int a) { return a*a; }
int main()
{
  // pointer to function that returns int
  int (*pf)(int);
  pf = sqr;
  printf("%d␣␣%d\n", sqr(4), pf(4));
  return 0;
}
```

```c
#include<stdio.h>
int f1() { return 1; }
int f2() { return 2; }
int f3() { return 3; }
typedef int(*pf)();
int main()
{
  int i;
  pf F[3];
  // F is an array of pointers to functions
  // returning int with argument int
  F[0] = f1;
  F[1] = f2;
  F[2] = f3;
  for (i=0; i<3; i++)
    printf("f%d␣returns␣␣%d\n", i, F[i]() );
  return 0;
}
```

Arrays of pointers

- Allocate an $m \times n$ matrix
- Generate random entries
- Print them
- Deallocate the matrix

```c
#include<stdlib.h>
#include<stdio.h>
#include<assert.h>
int main()
{
  int m = 3, n = 4;
  int i,j;
  // pointer to pointer
  double **A;
  // allocate matrix
  A = malloc(sizeof(double*)*m);
  assert(A);
  for (i=0; i<m; i++){
    A[i] = malloc(sizeof(double)*n);
    assert(A[i]);
  }
```

```
  // initialize
  for (i=0; i<m; i++)
    for (j=0; j<n; j++) A[i][j] = drand48();
  // ouput
  for (i=0; i<m; i++){
    for (j=0; j<n; j++) printf("% .4f ", A[i][j]);
    printf("\n");
  }
  // deallocate
  for (i=0; i<m; i++) free(A[i]);
  free(A);
  return 0;
}
```

A better way to implement the above is

```
#include<stdlib.h>
#include<stdio.h>
#include<assert.h>
int main(){
  int m = 3, n = 4, i,j;
  double *A = malloc(sizeof(double)*m*n);
  assert(A);
  // initialize
  for (i=0; i<m; i++)
    for (j=0; j<n; j++) A[i*n+j] = drand48();
  // ouput
  for (i=0; i<m; i++){
    for (j=0; j<n; j++) printf("% .4f ", A[i*n+j]);
    printf("\n");
  }
  // deallocate
  free(A);
  return 0;
}
```

More examples

Can you explain the following code http://www.applied-mathematics.net/miniSSEL1BLAS/miniSSEL1BLAS.html

```
double  ddot_C(int n, double *x,double *y){
  double s=0.0;
  while (n--) {
    s+=*(x++) * *(y++); };
  return s;
}
float   sdot_C(int n, float  *x,float  *y){
  float  s=0.0;
  while (n--) {
    s+=*(x++) * *(y++); };
  return s;
}
void   saxpy_C(int n, float   a,float  *x, float  *y){
  while (n--)  *(x++)+=  a* *(y++); }
void   daxpy_C(int n, double  a,double *x, double *y){
  while (n--)  *(x++)+=  a* *(y++); }
```

. . . and this one

```
int i=n>>3, j=0, k=n&7;
double s=0.0;
while (i--){
    s+=a[j  ]*b[j  ];
    s+=a[j+1]*b[j+1];
    s+=a[j+2]*b[j+2];
    s+=a[j+3]*b[j+3];
    s+=a[j+4]*b[j+4];
    s+=a[j+5]*b[j+5];
    s+=a[j+6]*b[j+6];
    s+=a[j+7]*b[j+7];
    j+=8;
}
switch(k){
    case 7: s+=a[j]*b[j]; j++;
    case 6: s+=a[j]*b[j]; j++;
    case 5: s+=a[j]*b[j]; j++;
    case 4: s+=a[j]*b[j]; j++;
    case 3: s+=a[j]*b[j]; j++;
    case 2: s+=a[j]*b[j]; j++;
    case 1: s+=a[j]*b[j]; j++;
}
```