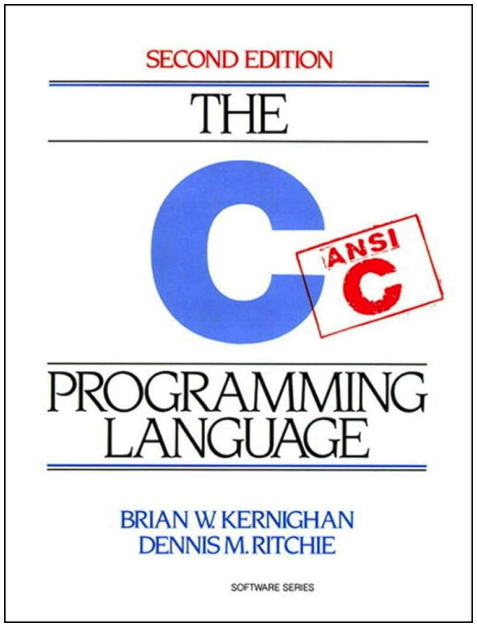
BASIC SYNTAX of ANSI C
Lecture Notes
BM 531
Numerical Methods and C/C++ Programming

Ahmet Ademoglu, *PhD*
Bogazici University
Institute of Biomedical Engineering



Simplest C code examples

```
/* Anything in between these slash-asterisk couple is treated as Comment */
#include <stdio.h>
int main(){
printf("Hello World\n");
return 0;
}
```

```
#include <stdio.h>
#define OutputMessage "Hello World\n"
int main(){
printf(OutputMessage);
return 0;
}
```

Declaration and Definition of Variables

```
int x=1;
float num;
int i,j,k;
double mean, variance;
char c;
unsigned m; /* or unsigned int : 4 bytes  */
unsigned char; /* 1 byte */
unsigned long; /* 8 byte */
unsigned short; /* 2 byte */
/* Array Definitions */
char string[80];
int image[256][256];
double volume_image[64][64][64];
/*  Structure Definitions  */
#include <string.h>
#include <stdio.h>
int main() {
struct student
{   int no;
char name[30];
char address[30];
int year;
} ;
struct student Ali;
strcpy(Ali.name, "Ali Ozay");
printf("Ali full name is %s",Ali.name);
return 0;
}
typedef struct student Student;
Student BM593Class[15];
BM593Class[0].name="Murat";
```

Pointers

```
int *p /* p is an address of an integer location */
*p = 1; /* Assignment */
```

p+1 denotes the address of the next integer location.

```
int x;
p=x /* p points to x */
x=*p /* copies the value of p to x */

int z[10];
p=&z[0]; /* p points to z[0] */
```

Relations between Pointers and Arrays

```
int a[10];
int *p;
p=&a[0];  or p=a;  /* p points the beginning address of a */
x=*p; /* the content of a[0] is copied into x */
x=*(p+1) /* the content of a[1] is copied into x */
x=a[i]; /* same as  x=*(a+1); */

p=a;  /* a=p is not allowed */
p++;  /* a++ is not allowed */
```

Statements

if

```
if (i==3)
x[i]+=x[i];

if (i<3)
x[i]+=3.;
else{
x[i]-=5.;
i--;
}
```

while

```
while (i > 0) {
x[i]=i*2.;
i++;
}
```

do while

```
do {
x[i]=i;
i--;
}while (i>4;)
```

Statements

switch

```
switch (i) {
case 'q' : printf ("It is quitting \n");
break;
case 'r' : printf ("It is reading \n");
break;
break : printf ("It is doing nothing \n");
}
```

break

```
for (i=0; i<5; i++)
if (x[i]>3)
break;
```

continue

```
for (i=0,sum=0; i<5; i++){
if (i>3) continue;
sum+=i;
}
```

for

```
for (i=0; i<10; i++) x[i]=3;
for (i=0; i<10; i++){
      for (j=0; j<10; j++)
      a[i][j]=i+2*j;
      x[i]=i; };
for (i=0,k=0; i<10;i++,k++)  x[i]=i;
```

Dynamic Memory Allocation

Vector allocation

```
#include <stdlib.h>
void main(){

int *int_vector;
int_vector = (int *) malloc(10*sizeof(int));  /* or */
/* int_vector = (int *) calloc(10,sizeof(int)); */
/* to free the dynamic memory */
free (int_vector);
}
```

Matrix allocation

```
#include <stdlib.h>
void main(){

int **int_matrix,i;
int_matrix = (int **) malloc(10*sizeof(int*));  /* or */
/* int_matrix = (int **) calloc(10,sizeof(int*)); */
for (i=0;i<10; i++)
int_matrix[i] = (int *) malloc(20*sizeof(int)); /* or */
/* for (i=0;i<10; i++)
int_matrix[i] = (int *) calloc(20*sizeof(int)); */
/* to free the dynamic memory */
for (i=0;i<10; i++)
free (int_matrix[i]);
free(int_matrix);
}
```

Functions

```
#include <stdio.h>
int power (int , int )
int main(){
int i;
for (i=0; i<10; i++)
printf ("%d %d\n", i, power(2,i)); return 0;
}
int power (int m, int n)
{
int i,p;
p=1;
for (i=1; i<=n; i++)
p*=m;
return p;
}
#include <stdio.h>
void power (int , int , int *)
int main(){
int i,p;
for (i=0; i<10; i++){
power(2,i,&p);
printf ("%d %d\n", i,p);
} ; return 0;  }
void  power (int m, int n, int *p){
int i;
*p=1;
for (i=1; i<=n; i++)
*p*=m;}
```

Passing Arrays to Functions

```
#include <stdlib.h>

int main(){

int i,**int_matrix;

int_matrix = (int **) calloc(10,sizeof(int*));
for (i=0;i<10; i++)
int_matrix[i] = (int *) malloc(20*sizeof(int));

scaleImage (int_matrix, 100, 10, 20);

/* to free the dynamic memory */
for (i=0;i<10; i++)
free (int_matrix[i]);
free(int_matrix);
}

void scaleImage (int ** int_matrix, int scale, int row, int column){
int i,j;

for (i=0;i<row,i++)
for (j=0;j<column;j++)
int_matrix[i][j]*=scale;
return 0;
}
```

Standard Input/Output

```
#include <stdio.h>

void main(){
long unsigned int i,*p;

printf("Enter the value of x\n");
scanf("%lu",&x);
printf("the value of x is %lu\n",x);

printf("Enter the value of p\n");
scanf("%lu",p);
printf("the value of p is %lu\n",*p);
}
```

File Input/Output

File Input/Output for ASCII TEXT FILES

```
#include <stdio.h>
void main(){
FILE *f1;
FILE *f2;
double x[100];
f1=fopen("c:\\data\\input.dat","r");
f2=fopen("c:\\data\\output.dat","w");
for (i=0;i<100;i++){
fscanf(f1,"%lf",&x[i]); /* or fscanf(f1,"%lf",x+i) */
fprintf(f2,"%lf",x[i]); /* or fprintf(f2,"%lf",*(x+i)) */
};
fclose(f1);
fclose(f2);
}
```

File Input/Output for BINARY DATA FILES

```
#include <stdio.h>
void main(){
FILE *f1;
FILE *f2;
double x[100];
f1=fopen("c:\\data\\input.dat","r+b");
f2=fopen("c:\\data\\output.dat","w+b");
for (i=0;i<100;i++){
fread(x,sizeof(double),100,f1);
fwrite(x,sizeof(double),100,f2);
};
```

Arithmetic Operators

```
int x,y;

x=1;
x+=3;
x=y-3;
x=x*y;
x/y;
x%y;
++x;
x++;
--x;
--y;
y=-x;
```

Relational and Logical Operators

```
(x>y) /* 1 if x greater than y else 0 */
(x>=y) /* 1 if x greater than or equal to y else 0 */
(x<y) /* 1 if x less than y else 0 */
(x<=y) /* 1 if x less than or equal to y else 0 */
(x0=y) /* 1 if x equals y else 0 */
(x!=y) /* 1 if x is not equal y else 0 */
(!x)   /* 1 if x is  equal 0 else 0 */
(x&&y) /* 1 if both x and y are 1 else 0 */
(x||y) /* 1 if either x or y or both are 1 else 0 */
z ? x:y /** If z is 1 than evaluate x else evaluate y  */
Ex: if (x>y) ? (max=x) : (max=y); /* Finds whichever is greater between x and y  */
(type) x /* changes the value of x to type */
Ex: int x=2; float y; y = (float) x /* Finds whichever is greater between x and y  */
```

Bitwise Operators

```
~x /* Complements all bits */
x&y /* bitwise AND */
x|y /* bitwise OR */
x^y /* bitwise EXCLUSIVE OR */
x>>y /* shift to right by y bits or divide by 2^y */
x<<y /* shift to left by y bits or multiply by 2^y */
```

File Operations

```
fseek(f1,100); /* puts the current file pointer 100
bytes ahead of the beginning of the file */
rewind(f1); /* sets the current file pointer to the beginning of the file
when some data has already been written to or read from the file */
```

Command Line Arguments

```
void main(int argc, char *argv[]){
/* File Copy */
char c;
FILE *f1,*f2;
if (argc == 3) {
f1=fopen(*++argv,"r");
f2=fopen(*++argv,"w");
c=getc(f1);
while (c ==!EOF){
putc(c,f2);
c=getc(f1);
}
fclose(f1);
fclose(f2);
}
else
printf("Usage: copy inputfile outputfile\n");    }
```

Macro Substitution for fast exection

```
#define max(A,B) ((A)>(B) ? (A) : (B))

#define SQR(A) ((A)*(A))
#define EUCLID(A,B) ( sqrt( (SQR(A))+(SQR(B)) ) )

#define SIGN(A,B) ( (B) > 0 ? fabs(A) : -fabs(A))

#define SWAP(A,B) {float temp=(A);(A)=(B);(B)=temp;}

#define IGREG (15+31L*(12L+1582))
```

Function Pointers

```
#include <stdio.h>
int trapezoid (*func)(int), int, int);
int myfunction (int);
int yourfunction (int);
void main(int argc, char *argv[]){
int x=1;
int y=2;
printf ("My trapezoidal value is %d\n",trapezoid(myfunction,x,y));
printf ("Your trapezoidal value is %d\n",trapezoid(yourfunction,x,y));
}
int trapezoid(int (*func)(int), int a, int b){
return ( (a+b)*(*func)(3)) ;
}
int myfunction(int x){
return (1+x)
}
int yourfunction(int x){
return (1+3*x)
}
```

qsort function call by binding function pointers

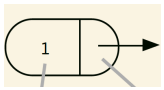
```
#include <stdio.h>
#include <stdlib.h>  /* qsort is defined in stdlib.h */
#define IMAXVALUES 10
int icompare_func(const void *, const void *);
int (*ifunct_ptr)(const void *, const void *);
void main(){
int i;
int iarray[IMAXVALUES]={0,5,3,2,8,7,9,1,4,6};
ifunct_ptr=icompare_funct;
qsort(iarray,IMAXVALUES,sizeof(int),ifunct_ptr);
for (i=0; i<IMAXVALUES;i++)  printf("%d ",iarray[i]);  }
int icompare_funct(const void *iresult_a, const void *iresult_b){
return ((*(int *)iresult_a) - (*(int *) iresult_b) ); }
```

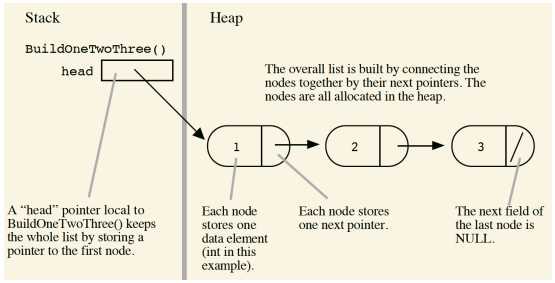
Other Examples of Function Pointers

```
int *(*(*ifunct_ptr)(int))[5];
/* a function pointer to a function that passed
an integer argument and returns a pointer to
array of 5 int pointers*/
float (*(*ffunct_ptr)(int,int))(float);
/* a function pointer to a function that takes two arguments and
returns a pointer to
a function taking a float argument and returning a float*/
typedef double (*(*(*dfunct_ptr)())[5])();
/* dfunct_ptr() is defined as a pointer to a function that is
passed nothing and returns
a pointer to an array of 5 pointers that point to functions that are
passed nothing and return  double */
dfunct_ptr A_dfunct_ptr;
int (*(*function_ary_ptrs())[5])();
/* a function declaration function_ary_ptrs() that takes no arguments and returns a
pointer to an array of five pointers that point to functions
taking no arguments and returning integers */
```

Linked Lists

```
struct node {
    int data;
    struct node* next;
}
```





Drawing of List {1,2,3}

```
struct node* Build123(){
struct node* head=NULL;
struct node* second=NULL;
struct node* third=NULL;
head = malloc(sizeof(struct node));
second = malloc(sizeof(struct node));
third = malloc(sizeof(struct node));
head->data=1;
head->next=second;
second->data=2;
second->next=third;
third->data=3;
third->next=NULL;
return head;
}
```

How can you write the code with only 7 assignments (=)?

```
#include<stdio.h>
#include<stdlib.h>
struct node {
int data;
struct node* next;
};
struct node* Build123(){
struct node* head = (struct node*) malloc(sizeof(struct node));
head->data=1;
head->next=(struct node*)  malloc(sizeof(struct node));
head->next->data =2;
head->next->next=(struct node*)  malloc(sizeof(struct node));
head->next->next->data =3;
head->next->next->next = NULL;
return head;
}
int  main(){
struct node* head = Build123();
while (head !=NULL){
printf("%d \n",head->data);
head = head->next;
}
return 0;
}
```

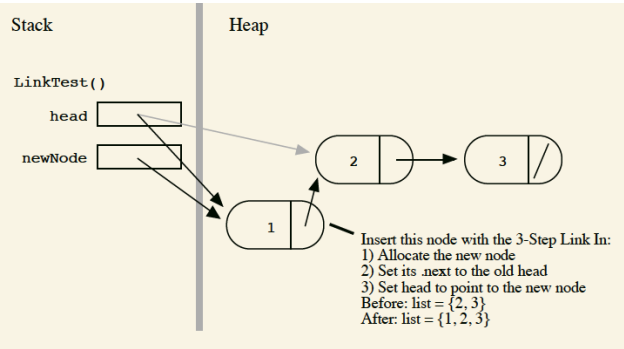
Given a linked list head pointer, compute and return the number of nodes in the list.

The Length() function takes a linked list and computes the number of elements in the list.

```
...
int  Length(struct node* head) {
struct node* current = head;
int count = 0;
while (current != NULL) {
count++;
current = current->next;
}
return count;
}
int  main(){
struct node* head = Build123();
printf("The length of the list is %d \n",Length(head));
return 0;
}
```

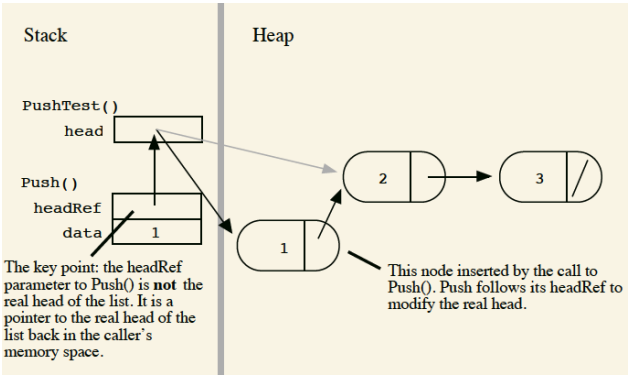
List Building

```
void LinkTest(){
struct node* head = Built23();// suppose this builds the {2, 3} list
struct node* newNode;
newNode=malloc(sizeof(struct node));
newNode->data=1;
newNode->next = head;
head = newNode;
}
```



Push() Function

```
void WrongPush (struct node* head, int data){
struct node* newNode = malloc(sizeof(struct node));
newNode->data=data;
newNode->next=head;
head=newNode;
}
void WrongPushTest(){
struct node* head = BuildTwoThree();
WrongPush(head, 1);
} /* WONT WORK */
void Push(struct node** headRef, int data){
struct node* newNode =malloc(sizeof(struct node));
newNode->data=data;
newNode->next=*headRef;
*headRef=newNode;
}
void PushTest(){
struct node* head=Built23();
Push(&head,1);
Push(&head,13);
} /* WILL WORK */
```



AppendNode() function is like Push(), except it adds the new node at the tail end of the list instead of the head.

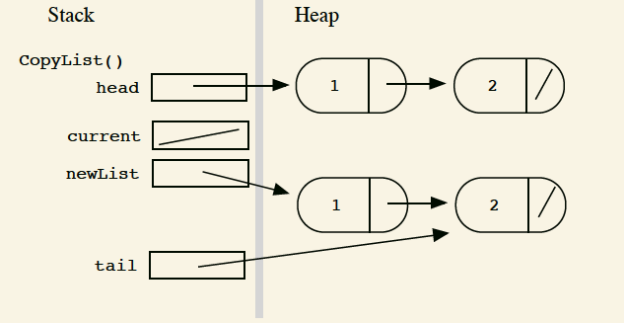
```
struct node* AppendNode(struct node** headRef, int num){
struct node* current =*headRef;
struct node* newNode;
newNode=malloc(sizeof(struct node));
newNode->data=num;
newNode->next=NULL;
if (current==NULL)
*headRef=newNode;
else{
while (current->next!=NULL)
current=current->next;
current->next=newNode;
}
}
```

AppendNode() With Push()

```
struct node* AppendNode(struct node** headRef, int num){
struct node* current =*headRef;
if (current==NULL)
Push(headRef,num);
else {
while (current->next!=NULL)
current=current->next;
Push(&(current->next),num); }
}
```

CopyList() Example CopyList() function takes a list and returns a complete copy of that list.

```
struct node* CopyList(struct node* head) {
struct node* current = head;
// used to iterate over the original list
struct node* newList = NULL; // head of the new list
struct node* tail = NULL;
// kept pointing to the last node in the new list
while (current != NULL){
if (newList == NULL) {     // special case for the first new
nodenewList = malloc(sizeof(struct node));
newList->data = current->data;
newList->next = NULL;
tail = newList;   }
else {
tail->next = malloc(sizeof(struct node));
tail = tail->next;
tail->data = current->data;
tail->next = NULL;
}  current = current->next;
}  return(newList);  }
```



CopyList() with Push()

```
struct node* CopyList2(struct node* head){
struct node* current =head;
struct node* newlist =NULL;
struct node* tail =NULL;
while (current!=NULL){
if (newlist==NULL){
Push(&newlist,current->data);
tail=newlist;
}
else {
Push(&(tail->next),current->data);
tail=tail->next;
}
current=current->next;
}
return(newlist);
}
```

Source Code for Creating a List using typedef

```
#include <stdlib.h>
#include <stdio.h>
struct node {
int data;
struct node* next;};
typedef struct node nd;
nd* CopyList(nd* head){
if (head==NULL) return NULL;
else {
nd* newlist=malloc(sizeof(nd));
newlist->data=head->data;
newlist->next=CopyList(head->next);
return(newlist);}}
nd* Build123(){
nd* head = (nd*) malloc(sizeof(nd));
head->data=1;
head->next=(nd*)  malloc(sizeof( nd));
head->next->data =2;
head->next->next=(nd*)  malloc(sizeof(nd));
head->next->next->data =3;
head->next->next->next = NULL;
return head;}
int  main(){
nd* head = Build123();
nd* pt = head;
while ( pt !=NULL){  printf("%d \n",pt->data); pt = pt->next; }
nd* copyhead = CopyList(head);
while (copyhead !=NULL){
printf("%d \n",copyhead->data);
copyhead = copyhead->next; }
return 0;}
```

CopyList() With Dummy Node

```
// Dummy node variant
struct node* CopyList(struct node* head) {
struct node* current = head; // used to iterate over the original list
struct node* tail; // kept pointing to the last node in the new list
struct node dummy; // build the new list off this dummy node
dummy.next = NULL;
tail = &dummy; // start the tail pointing at the dummy
while (current != NULL) {
Push(&(tail->next), current->data); // add each node at the tail
tail = tail->next; // advance the tail to the new last node
current = current->next;
}
return(dummy.next);
}
```

CopyList() Recursive

```
struct node* CopyList(struct node* head){
if (head==NULL) return NULL;
else {
struct node* newlist=malloc(sizeof(struct node));
newlist->data=head->data;
newlist->next=CopyList(head->next);
{
printf("%d\n",newlist->data);
return(newlist);
}
}
}
```

Sample Program for CopyList() Recursive

```
#include <stdlib.h>
#include <stdio.h>
struct node {
int data;
struct node* next;};
typedef struct node nd;
nd* CopyList(struct node* head){
if (head==NULL) return NULL;
else {
nd* newlist=malloc(sizeof(nd));
newlist->data=head->data;
newlist->next=CopyList(head->next);{ return(newlist);}}}
nd* Build123(){
nd* head = (nd*) malloc(sizeof(nd));
head->data=1;
head->next=(nd*) malloc(sizeof( nd));
head->next->data =2;
head->next->next=(nd*) malloc(sizeof(nd));
head->next->next->data =3;
head->next->next->next = NULL;
return head;
}
int main(){
nd* head = Build123();
nd* pt = head;
while ( pt !=NULL){ printf("Original List %d \n",pt->data);
pt = pt->next; }
nd* copyhead = CopyList(head);
while (copyhead !=NULL){ printf("Copied List %d \n",copyhead->data);
copyhead = copyhead->next; }
}
```