List (1A)

Young Won Lim
4/4/15

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice and Octave.

Struct Declaration

structure type

```
struct aaa {
        int     i;
        short   s;
        char    c;
} ;
```

```
struct aaa  var;
```

structure type

```
struct aaa {
        int     i;
        short   s;
        char    c;
} ;
```

```
typedef struct aaa  ATYPE ;
```

```
ATYPE  var;
```

structure type

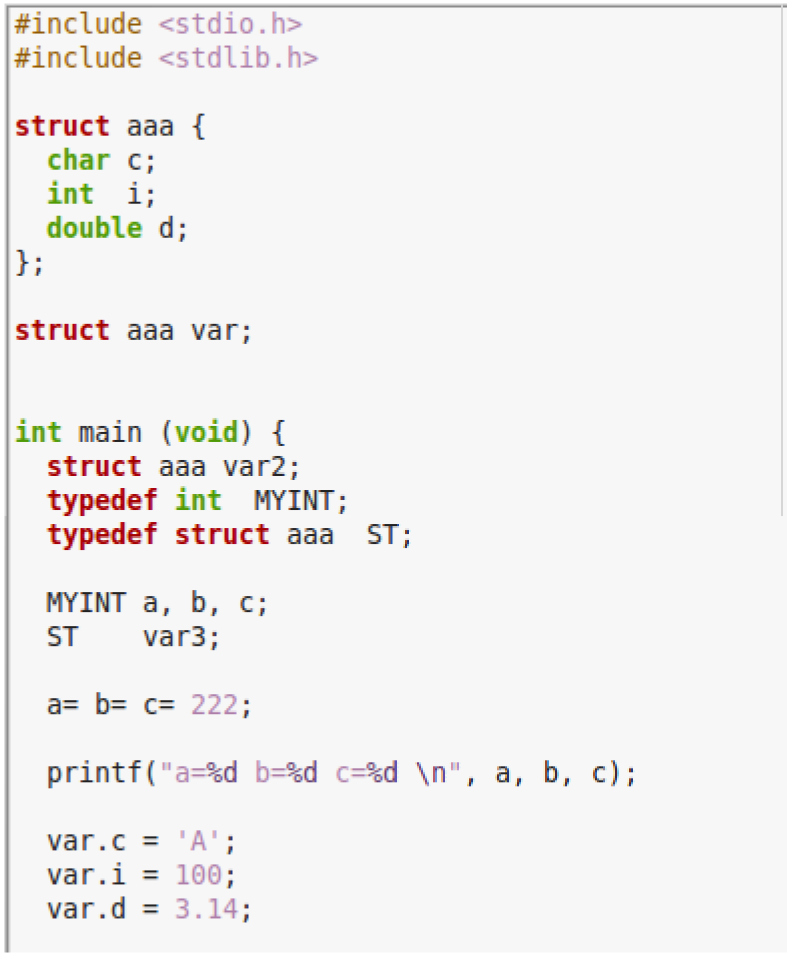
```
struct aaa {
        int     i;
        short   s;
        char    c;
} var ;
```

structure type

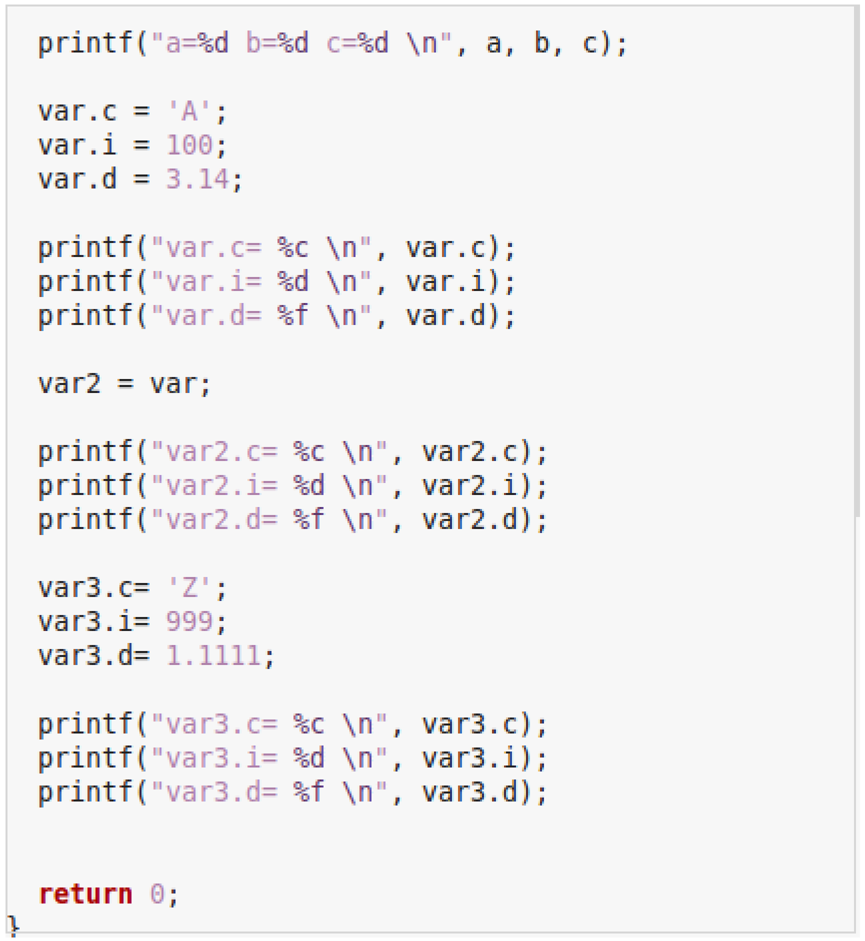
```
typedef struct aaa {
        int     i;
        short   s;
        char    c;
} ATYPE ;
```

```
ATYPE  var;
```

Struct Type (1)

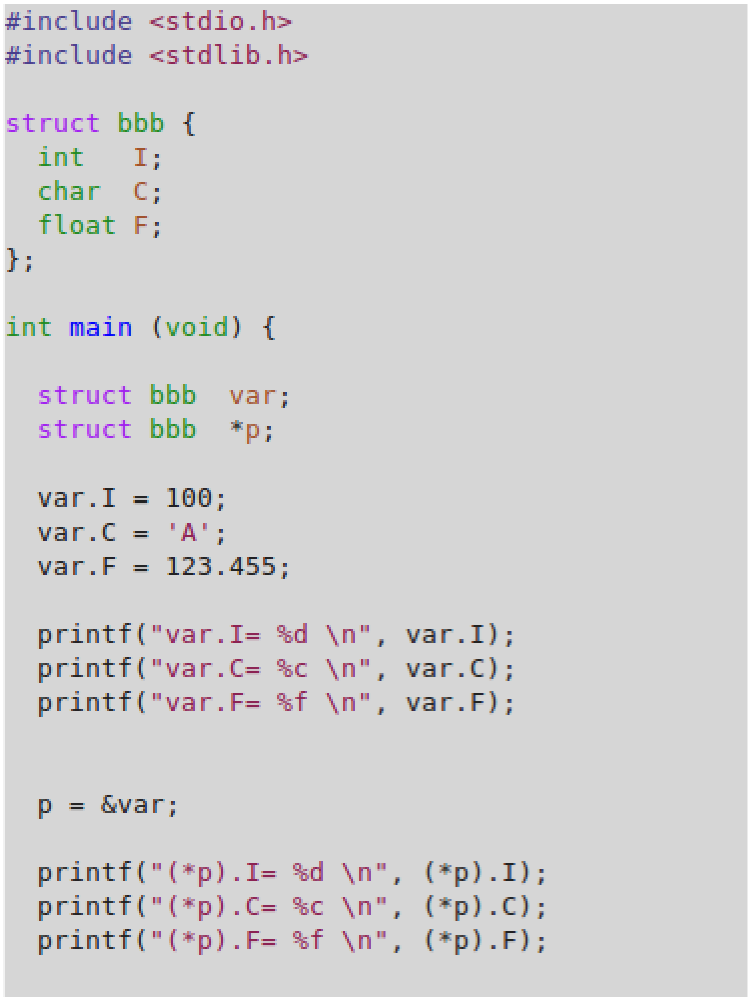


```
#include <stdio.h>
#include <stdlib.h>

struct aaa {
  char c;
  int  i;
  double d;
};

struct aaa var;


int main (void) {
  struct aaa var2;
  typedef int  MYINT;
  typedef struct aaa  ST;

  MYINT a, b, c;
  ST    var3;

  a= b= c= 222;

  printf("a=%d b=%d c=%d \n", a, b, c);

  var.c = 'A';
  var.i = 100;
  var.d = 3.14;
```

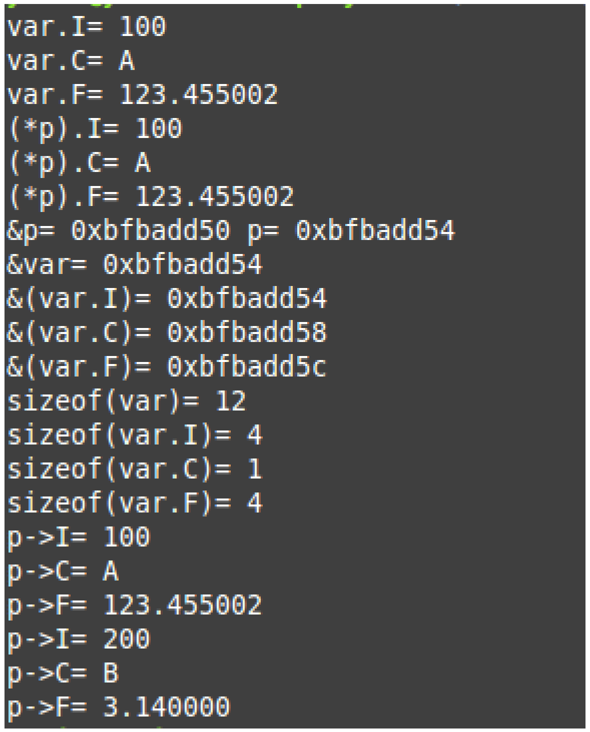


```
  printf("a=%d b=%d c=%d \n", a, b, c);

  var.c = 'A';
  var.i = 100;
  var.d = 3.14;

  printf("var.c= %c \n", var.c);
  printf("var.i= %d \n", var.i);
  printf("var.d= %f \n", var.d);

  var2 = var;

  printf("var2.c= %c \n", var2.c);
  printf("var2.i= %d \n", var2.i);
  printf("var2.d= %f \n", var2.d);

  var3.c= 'Z';
  var3.i= 999;
  var3.d= 1.1111;

  printf("var3.c= %c \n", var3.c);
  printf("var3.i= %d \n", var3.i);
  printf("var3.d= %f \n", var3.d);


  return 0;
}
```

Struct Type (2)

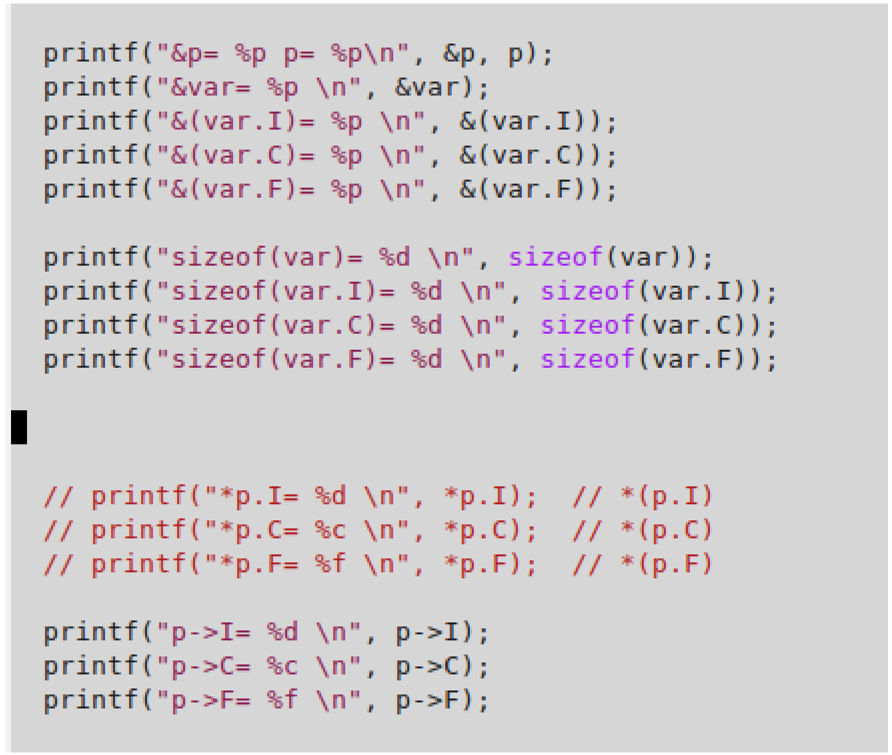


```c
#include <stdio.h>
#include <stdlib.h>

struct bbb {
  int   I;
  char  C;
  float F;
};

int main (void) {

  struct bbb  var;
  struct bbb  *p;

  var.I = 100;
  var.C = 'A';
  var.F = 123.455;

  printf("var.I= %d \n", var.I);
  printf("var.C= %c \n", var.C);
  printf("var.F= %f \n", var.F);


  p = &var;

  printf("(*p).I= %d \n", (*p).I);
  printf("(*p).C= %c \n", (*p).C);
  printf("(*p).F= %f \n", (*p).F);
```

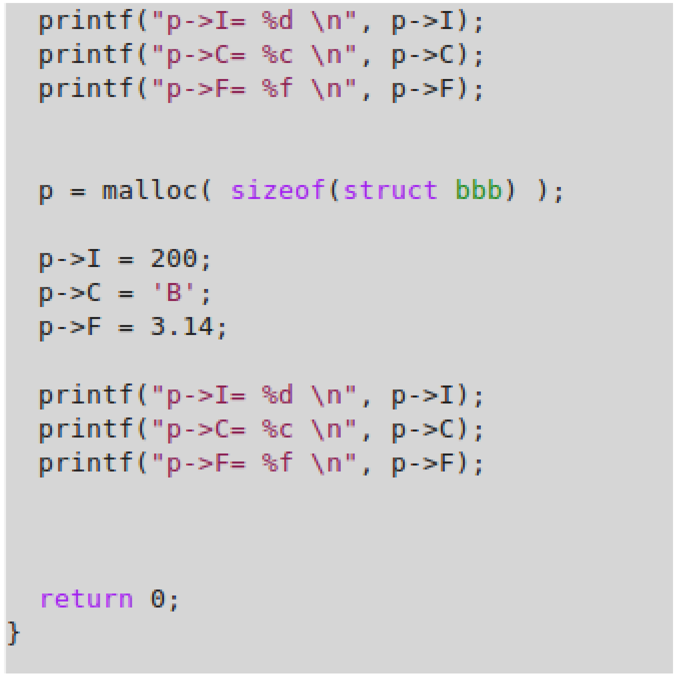


```
var.I= 100
var.C= A
var.F= 123.455002
(*p).I= 100
(*p).C= A
(*p).F= 123.455002
&p= 0xbfbadd50 p= 0xbfbadd54
&var= 0xbfbadd54
&(var.I)= 0xbfbadd54
&(var.C)= 0xbfbadd58
&(var.F)= 0xbfbadd5c
sizeof(var)= 12
sizeof(var.I)= 4
sizeof(var.C)= 1
sizeof(var.F)= 4
p->I= 100
p->C= A
p->F= 123.455002
p->I= 200
p->C= B
p->F= 3.140000
```

Struct Type (3)



```
printf("&p= %p p= %p\n", &p, p);
printf("&var= %p \n", &var);
printf("&(var.I)= %p \n", &(var.I));
printf("&(var.C)= %p \n", &(var.C));
printf("&(var.F)= %p \n", &(var.F));

printf("sizeof(var)= %d \n", sizeof(var));
printf("sizeof(var.I)= %d \n", sizeof(var.I));
printf("sizeof(var.C)= %d \n", sizeof(var.C));
printf("sizeof(var.F)= %d \n", sizeof(var.F));



// printf("*p.I= %d \n", *p.I);  // *(p.I)
// printf("*p.C= %c \n", *p.C);  // *(p.C)
// printf("*p.F= %f \n", *p.F);  // *(p.F)

printf("p->I= %d \n", p->I);
printf("p->C= %c \n", p->C);
printf("p->F= %f \n", p->F);
```



```
  printf("p->I= %d \n", p->I);
  printf("p->C= %c \n", p->C);
  printf("p->F= %f \n", p->F);


  p = malloc( sizeof(struct bbb) );

  p->I = 200;
  p->C = 'B';
  p->F = 3.14;

  printf("p->I= %d \n", p->I);
  printf("p->C= %c \n", p->C);
  printf("p->F= %f \n", p->F);



  return 0;
}
```

Struct Type (4)

```c
#include <stdio.h>
#include <stdlib.h>

struct bbb {
  int   I;
  float F;
  char  C;
  char  C1;
  char  C2;
  char  C3;

};

int main (void) {

  struct bbb  var;

  printf("sizeof(var)= %d \n", sizeof(var));
  printf("sizeof(var.I)= %d \n", sizeof(var.I));
  printf("sizeof(var.C)= %d \n", sizeof(var.C));
  printf("sizeof(var.F)= %d \n", sizeof(var.F));
  printf("sizeof(var.C1)= %d \n", sizeof(var.C1));
  printf("sizeof(var.C2)= %d \n", sizeof(var.C2));
  printf("sizeof(var.C3)= %d \n", sizeof(var.C3));


  return 0;
}
```

Linked List Type (1)

```
struct aaa  {
        int            Data;
        struct aaa     *Next;
} ;
```

```
typedef struct aaa   node
```

```
typedef struct aaa  {
        int            Data;
        struct aaa     *Next;
} node;
```

```
typedef struct node  {
        int            Data;
        struct node    *Next;
} node;
```

Linked List Type (2)

```
struct aaa {
        int             Data;
        struct aaa      *Next;
} ;
```

```
typedef struct aaa  node
```

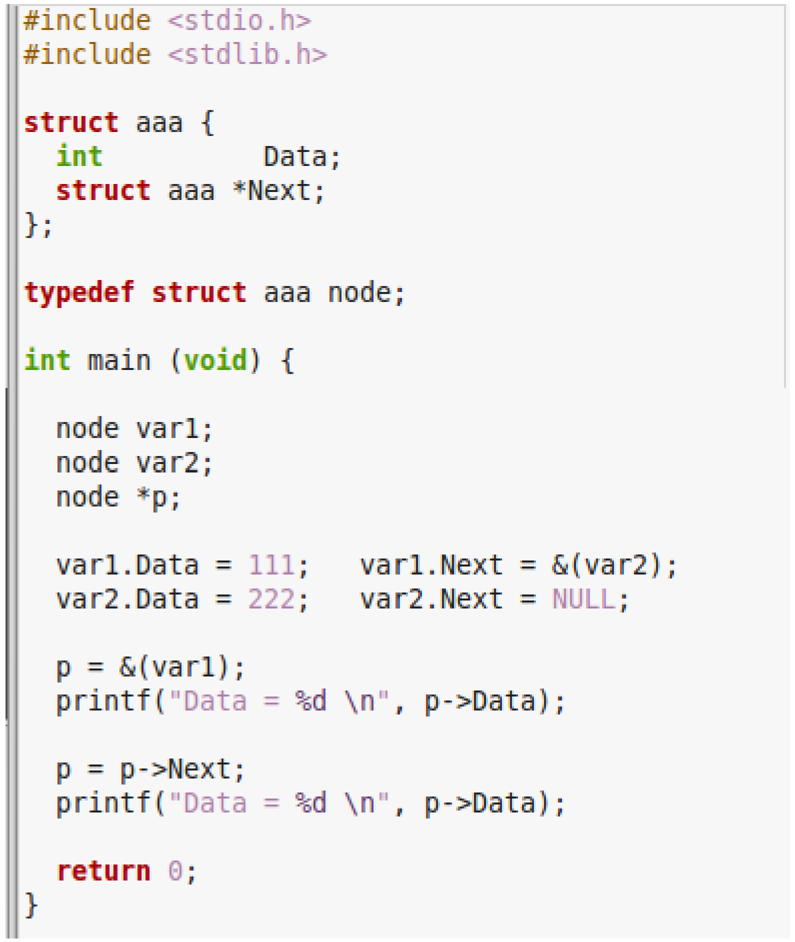
```
typedef struct aaa  node
```

```
struct aaa {
        int             Data;
        node            *Next;
} ;
```

```
typedef struct node node
```

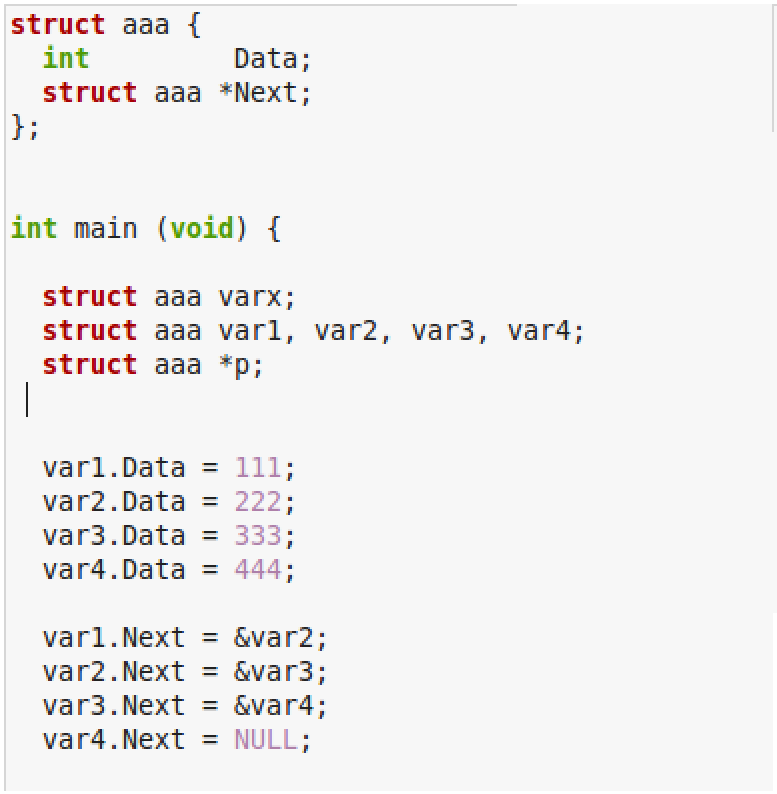
```
struct node {
        int             Data;
        node            *Next;
} ;
```

Simple List

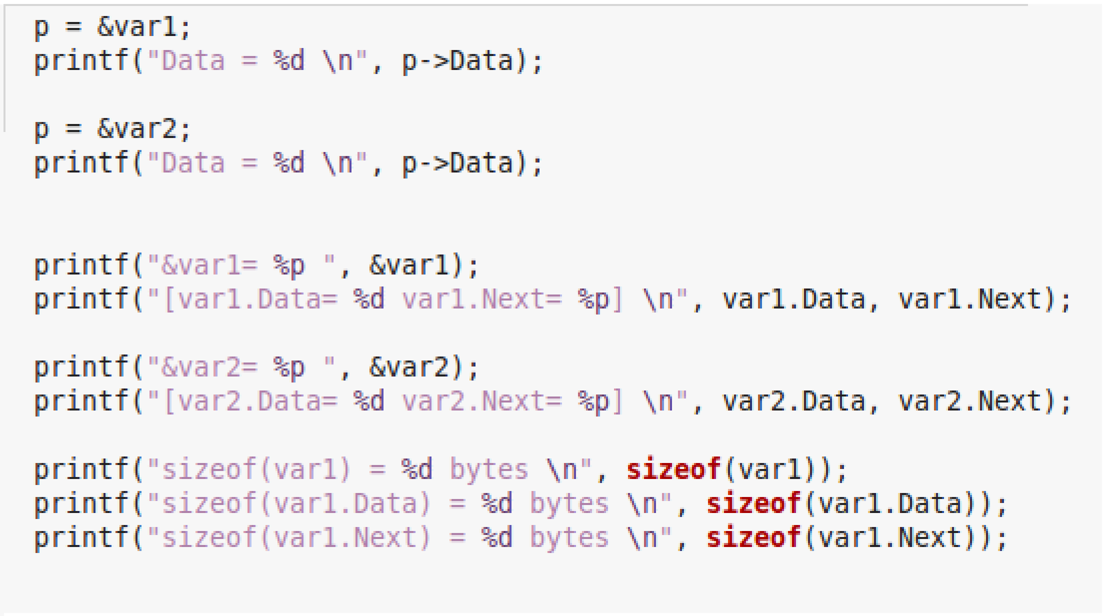


```c
#include <stdio.h>
#include <stdlib.h>

struct aaa {
  int          Data;
  struct aaa *Next;
};

typedef struct aaa node;

int main (void) {

  node var1;
  node var2;
  node *p;

  var1.Data = 111;   var1.Next = &(var2);
  var2.Data = 222;   var2.Next = NULL;

  p = &(var1);
  printf("Data = %d \n", p->Data);

  p = p->Next;
  printf("Data = %d \n", p->Data);

  return 0;
}
```

Traversing list elements (1)

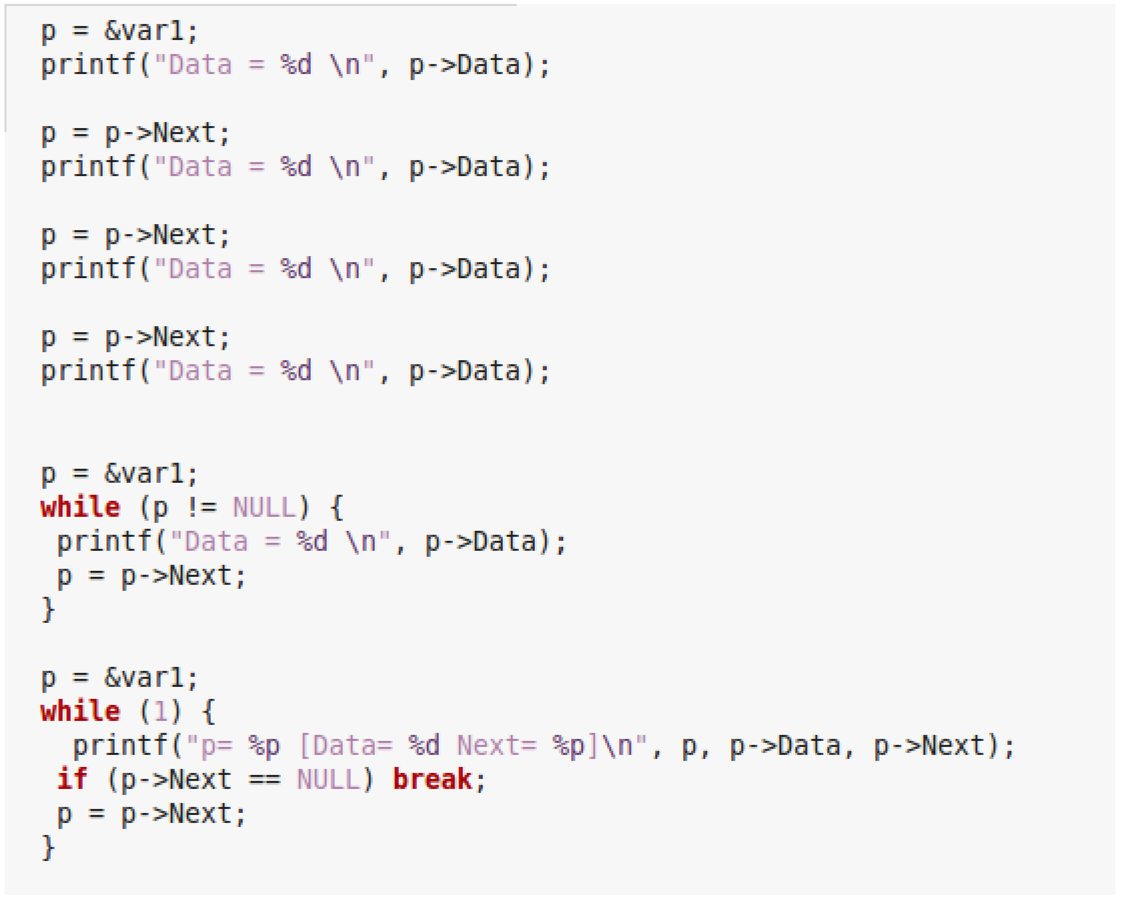


```c
struct aaa {
  int         Data;
  struct aaa *Next;
};


int main (void) {

  struct aaa varx;
  struct aaa var1, var2, var3, var4;
  struct aaa *p;


  var1.Data = 111;
  var2.Data = 222;
  var3.Data = 333;
  var4.Data = 444;

  var1.Next = &var2;
  var2.Next = &var3;
  var3.Next = &var4;
  var4.Next = NULL;
```



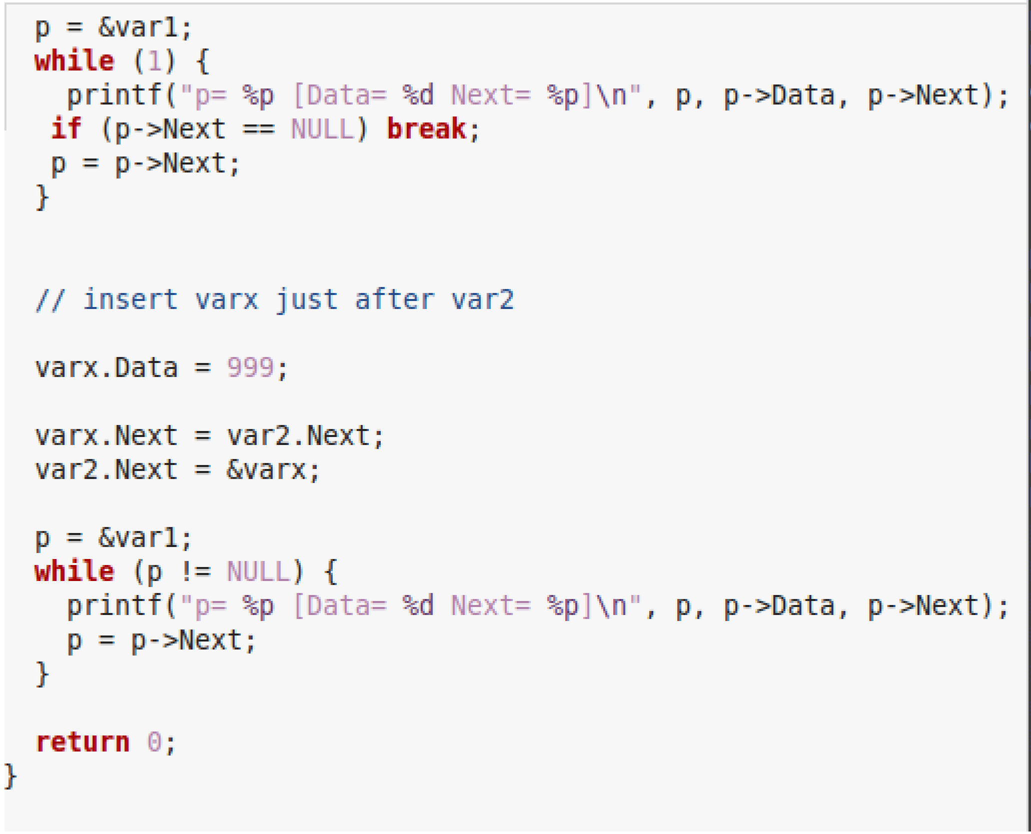
```c
  p = &var1;
  printf("Data = %d \n", p->Data);

  p = &var2;
  printf("Data = %d \n", p->Data);


  printf("&var1= %p ", &var1);
  printf("[var1.Data= %d var1.Next= %p] \n", var1.Data, var1.Next);

  printf("&var2= %p ", &var2);
  printf("[var2.Data= %d var2.Next= %p] \n", var2.Data, var2.Next);

  printf("sizeof(var1) = %d bytes \n", sizeof(var1));
  printf("sizeof(var1.Data) = %d bytes \n", sizeof(var1.Data));
  printf("sizeof(var1.Next) = %d bytes \n", sizeof(var1.Next));
```

Traversing list elements (2)

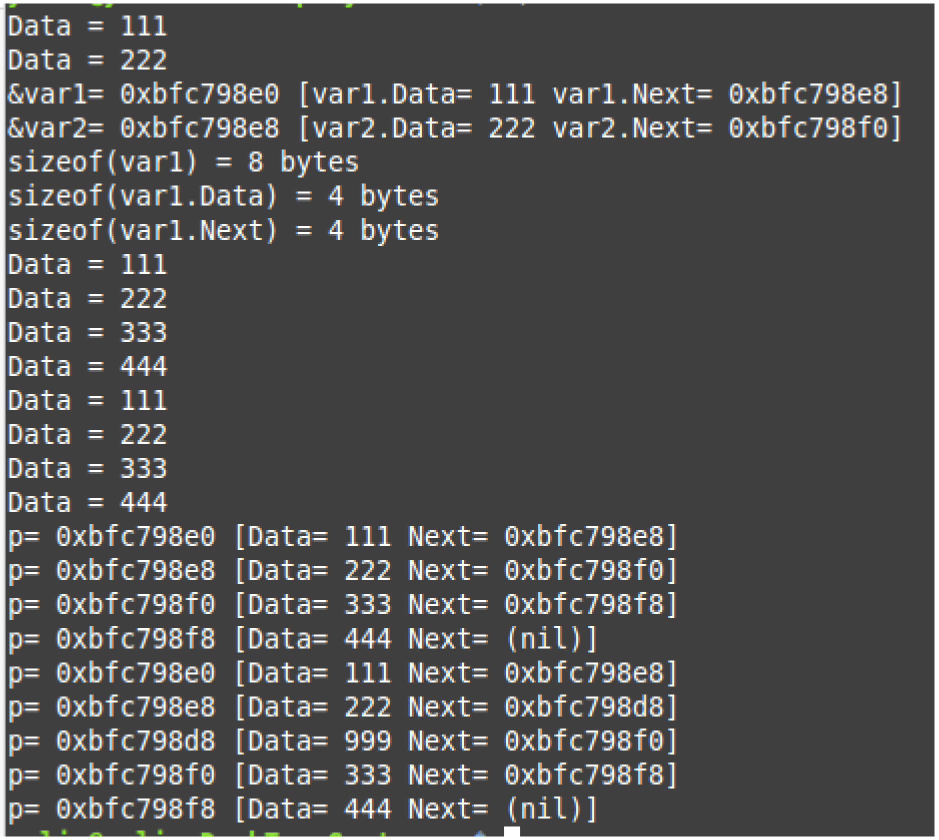


```c
p = &var1;
printf("Data = %d \n", p->Data);

p = p->Next;
printf("Data = %d \n", p->Data);

p = p->Next;
printf("Data = %d \n", p->Data);

p = p->Next;
printf("Data = %d \n", p->Data);


p = &var1;
while (p != NULL) {
 printf("Data = %d \n", p->Data);
 p = p->Next;
}

p = &var1;
while (1) {
  printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
 if (p->Next == NULL) break;
 p = p->Next;
}
```

Traversing list elements (3)

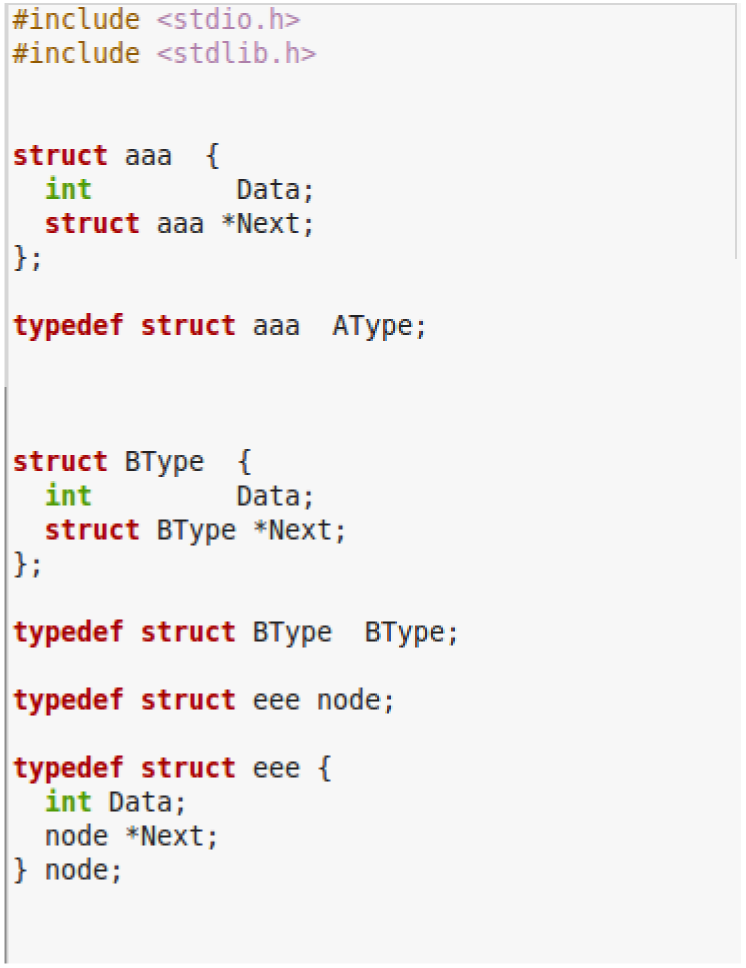


```c
  p = &var1;
  while (1) {
    printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
   if (p->Next == NULL) break;
   p = p->Next;
  }


  // insert varx just after var2

  varx.Data = 999;

  varx.Next = var2.Next;
  var2.Next = &varx;

  p = &var1;
  while (p != NULL) {
    printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
    p = p->Next;
  }

  return 0;
}
```

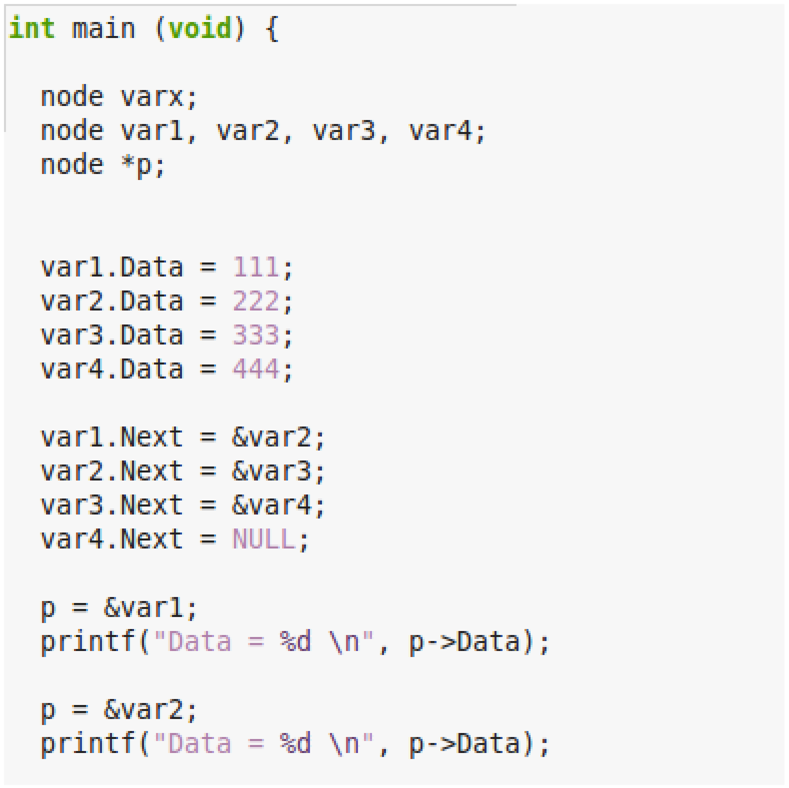


```
Data = 111
Data = 222
&var1= 0xbfc798e0 [var1.Data= 111 var1.Next= 0xbfc798e8]
&var2= 0xbfc798e8 [var2.Data= 222 var2.Next= 0xbfc798f0]
sizeof(var1) = 8 bytes
sizeof(var1.Data) = 4 bytes
sizeof(var1.Next) = 4 bytes
Data = 111
Data = 222
Data = 333
Data = 444
Data = 111
Data = 222
Data = 333
Data = 444
p= 0xbfc798e0 [Data= 111 Next= 0xbfc798e8]
p= 0xbfc798e8 [Data= 222 Next= 0xbfc798f0]
p= 0xbfc798f0 [Data= 333 Next= 0xbfc798f8]
p= 0xbfc798f8 [Data= 444 Next= (nil)]
p= 0xbfc798e0 [Data= 111 Next= 0xbfc798e8]
p= 0xbfc798e8 [Data= 222 Next= 0xbfc798d8]
p= 0xbfc798d8 [Data= 999 Next= 0xbfc798f0]
p= 0xbfc798f0 [Data= 333 Next= 0xbfc798f8]
p= 0xbfc798f8 [Data= 444 Next= (nil)]
```

Using type definition (1)

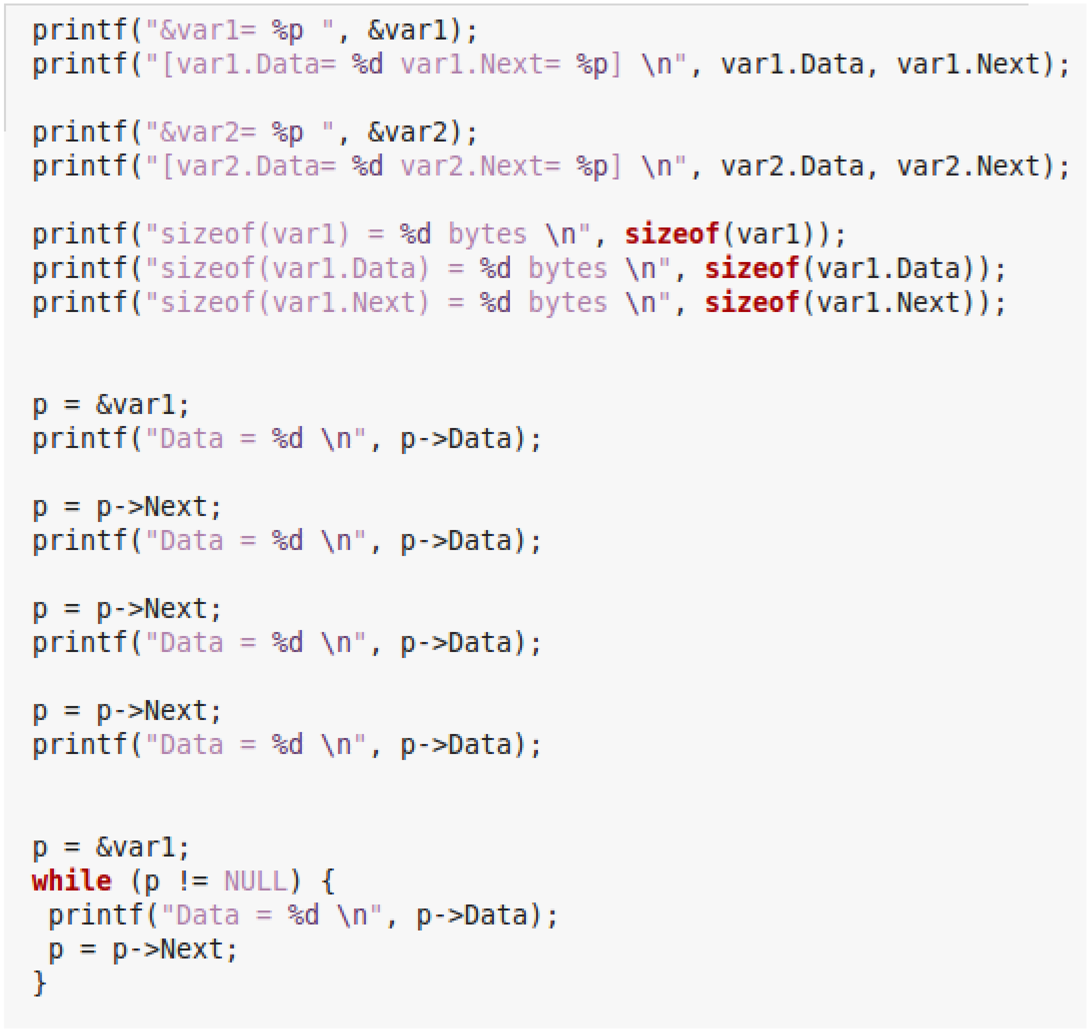


```c
#include <stdio.h>
#include <stdlib.h>


struct aaa  {
  int          Data;
  struct aaa *Next;
};

typedef struct aaa  AType;



struct BType  {
  int          Data;
  struct BType *Next;
};

typedef struct BType  BType;

typedef struct eee node;

typedef struct eee {
  int Data;
  node *Next;
} node;
```



```c
int main (void) {

  node varx;
  node var1, var2, var3, var4;
  node *p;


  var1.Data = 111;
  var2.Data = 222;
  var3.Data = 333;
  var4.Data = 444;

  var1.Next = &var2;
  var2.Next = &var3;
  var3.Next = &var4;
  var4.Next = NULL;

  p = &var1;
  printf("Data = %d \n", p->Data);

  p = &var2;
  printf("Data = %d \n", p->Data);
```

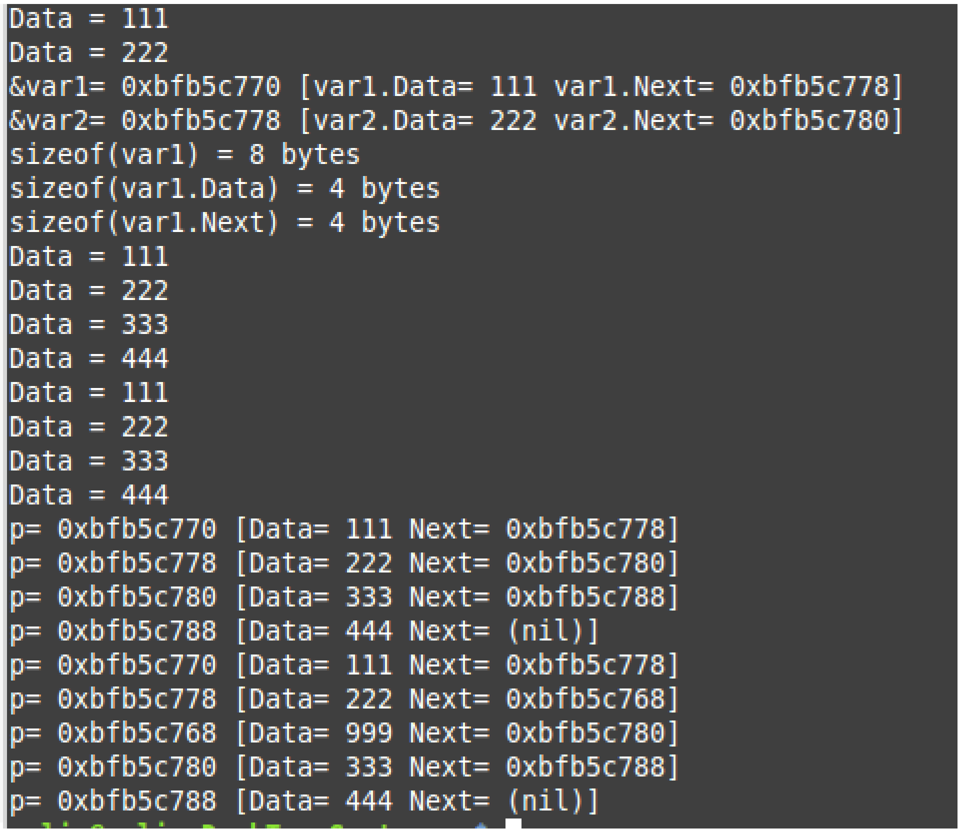
Using type definition (2)



```
printf("&var1= %p ", &var1);
printf("[var1.Data= %d var1.Next= %p] \n", var1.Data, var1.Next);

printf("&var2= %p ", &var2);
printf("[var2.Data= %d var2.Next= %p] \n", var2.Data, var2.Next);

printf("sizeof(var1) = %d bytes \n", sizeof(var1));
printf("sizeof(var1.Data) = %d bytes \n", sizeof(var1.Data));
printf("sizeof(var1.Next) = %d bytes \n", sizeof(var1.Next));


p = &var1;
printf("Data = %d \n", p->Data);

p = p->Next;
printf("Data = %d \n", p->Data);

p = p->Next;
printf("Data = %d \n", p->Data);

p = p->Next;
printf("Data = %d \n", p->Data);


p = &var1;
while (p != NULL) {
 printf("Data = %d \n", p->Data);
 p = p->Next;
}
```

Using type definition (3)

```c
  p = &var1;
  while (1) {
    printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
   if (p->Next == NULL) break;
   p = p->Next;
  }


  // insert varx just after var2

  varx.Data = 999;

  varx.Next = var2.Next;
  var2.Next = &varx;

  p = &var1;
  while (p != NULL) {
    printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
    p = p->Next;
  }

  return 0;
}
```



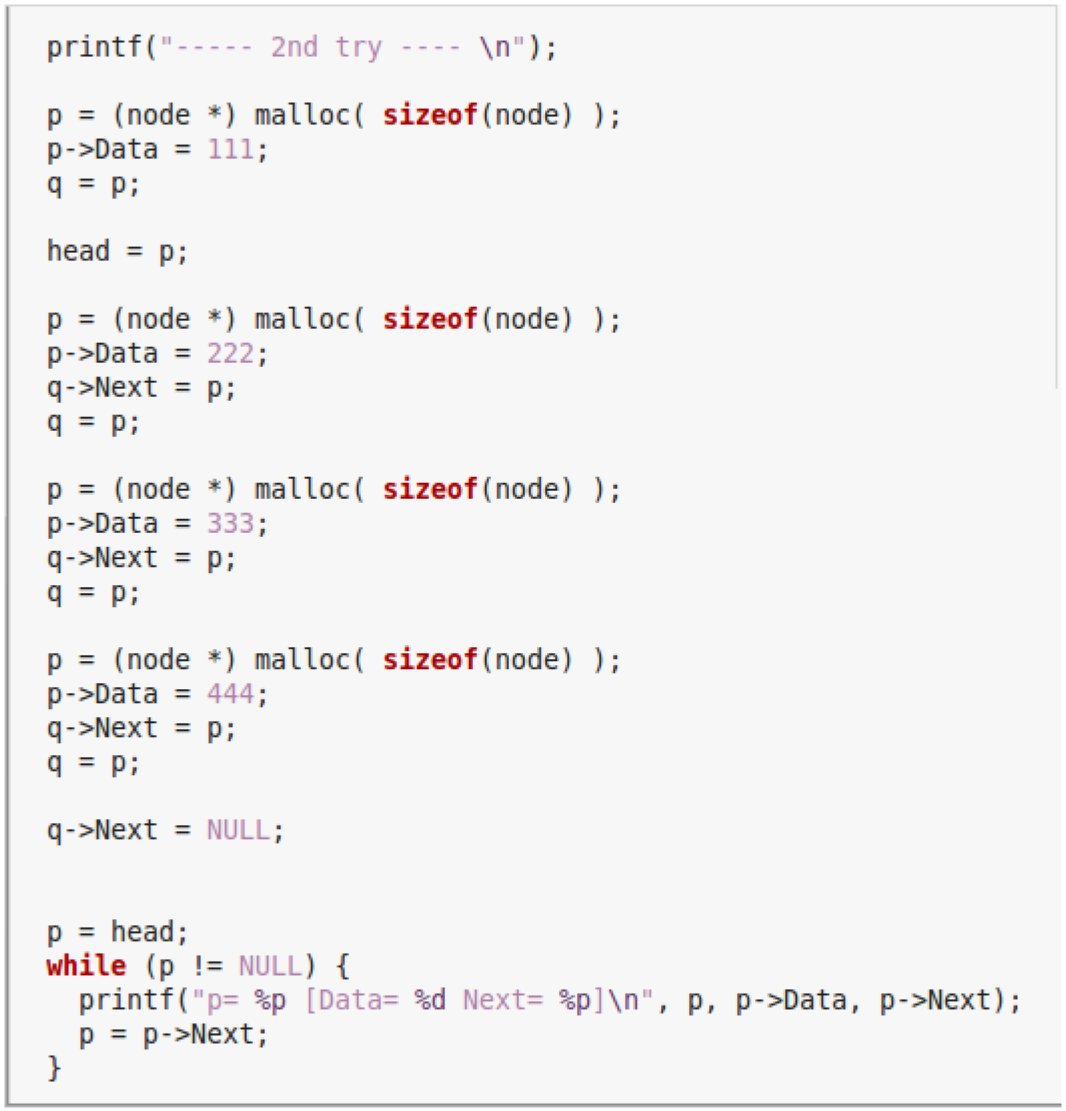
```
Data = 111
Data = 222
&var1= 0xbfb5c770 [var1.Data= 111 var1.Next= 0xbfb5c778]
&var2= 0xbfb5c778 [var2.Data= 222 var2.Next= 0xbfb5c780]
sizeof(var1) = 8 bytes
sizeof(var1.Data) = 4 bytes
sizeof(var1.Next) = 4 bytes
Data = 111
Data = 222
Data = 333
Data = 444
Data = 111
Data = 222
Data = 333
Data = 444
p= 0xbfb5c770 [Data= 111 Next= 0xbfb5c778]
p= 0xbfb5c778 [Data= 222 Next= 0xbfb5c780]
p= 0xbfb5c780 [Data= 333 Next= 0xbfb5c788]
p= 0xbfb5c788 [Data= 444 Next= (nil)]
p= 0xbfb5c770 [Data= 111 Next= 0xbfb5c778]
p= 0xbfb5c778 [Data= 222 Next= 0xbfb5c768]
p= 0xbfb5c768 [Data= 999 Next= 0xbfb5c780]
p= 0xbfb5c780 [Data= 333 Next= 0xbfb5c788]
p= 0xbfb5c788 [Data= 444 Next= (nil)]
```

Allocating list elements dynamically (1)

```
#include <stdio.h>
#include <stdlib.h>

typedef struct aaa node;

struct aaa {
  int    Data;
  node  *Next;
};


int main (void) {

  node *p1, *p2, *p3, *p4;
  node *p, *q, *head;


  printf("sizeof(node) = %d bytes \n", sizeof(node));
```

```
  printf("----- 1st try ---- \n");

  p1 = (node *) malloc( sizeof(node) );
  p1->Data = 111;

  p2 = (node *) malloc( sizeof(node) );
  p2->Data = 222;

  p3 = (node *) malloc( sizeof(node) );
  p3->Data = 333;

  p4 = (node *) malloc( sizeof(node) );
  p4->Data = 444;


  p1->Next = p2;
  p2->Next = p3;
  p3->Next = p4;
  p4->Next = NULL;


  p = p1;
  while (p != NULL) {
    printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
    p = p->Next;
  }
```

Allocating list elements dynamically (2)

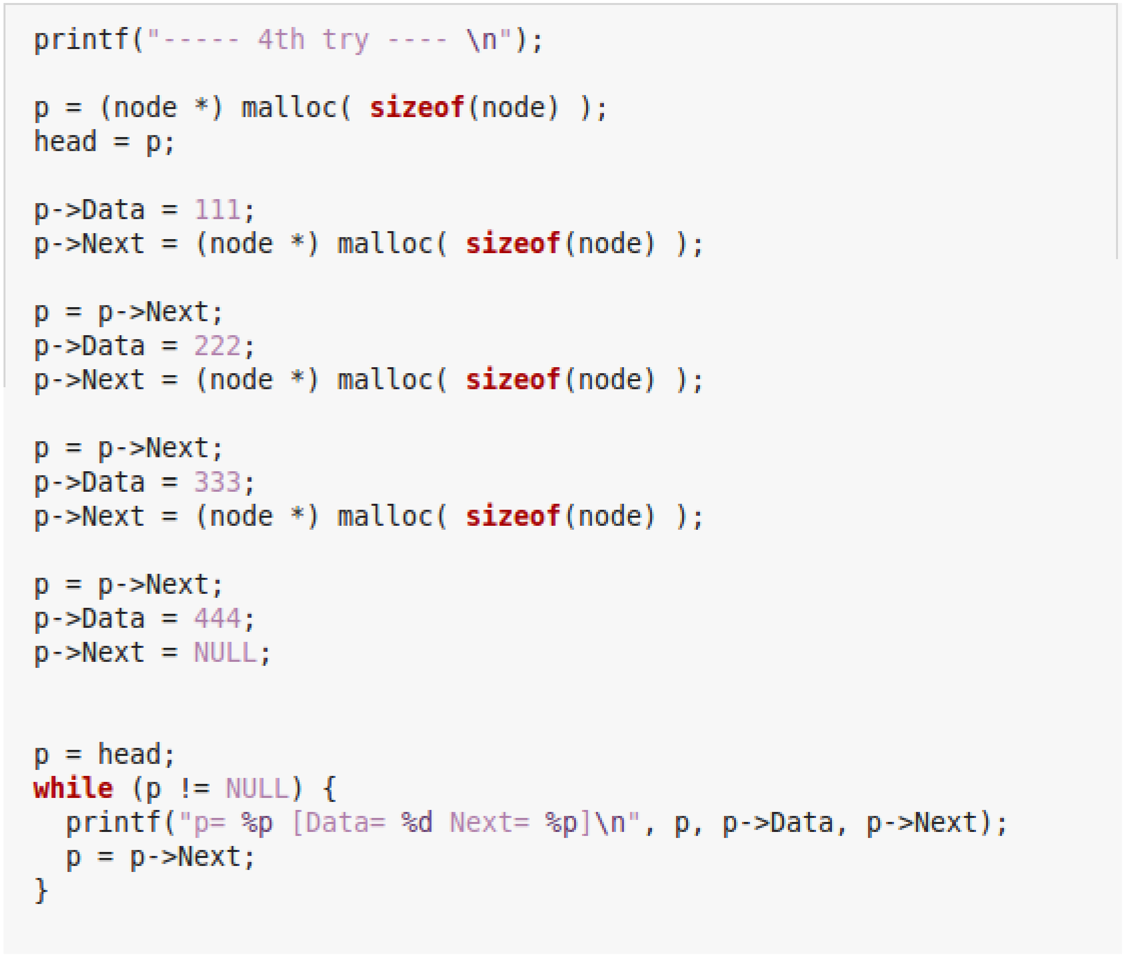


```c
printf("----- 2nd try ---- \n");

p = (node *) malloc( sizeof(node) );
p->Data = 111;
q = p;

head = p;

p = (node *) malloc( sizeof(node) );
p->Data = 222;
q->Next = p;
q = p;

p = (node *) malloc( sizeof(node) );
p->Data = 333;
q->Next = p;
q = p;

p = (node *) malloc( sizeof(node) );
p->Data = 444;
q->Next = p;
q = p;

q->Next = NULL;


p = head;
while (p != NULL) {
  printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
  p = p->Next;
}
```

Allocating list elements dynamically (3)

```
printf("----- 3rd try ---- \n");

p = (node *) malloc( sizeof(node) );
head = p;

p->Data = 111;
p->Next = (node *) malloc( sizeof(node) );

p->Next->Data = 222;
p->Next->Next = (node *) malloc( sizeof(node) );

p->Next->Next->Data = 333;
p->Next->Next->Next = (node *) malloc( sizeof(node) );

p->Next->Next->Next->Data = 444;
p->Next->Next->Next->Next = NULL;


p = head;
while (p != NULL) {
  printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
  p = p->Next;
}
```

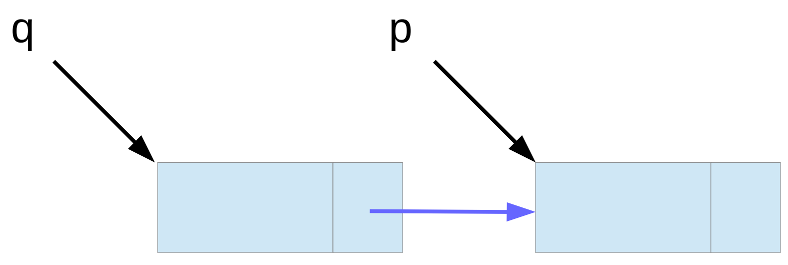
Allocating list elements dynamically (4)



```
printf("----- 4th try ---- \n");

p = (node *) malloc( sizeof(node) );
head = p;

p->Data = 111;
p->Next = (node *) malloc( sizeof(node) );

p = p->Next;
p->Data = 222;
p->Next = (node *) malloc( sizeof(node) );

p = p->Next;
p->Data = 333;
p->Next = (node *) malloc( sizeof(node) );

p = p->Next;
p->Data = 444;
p->Next = NULL;


p = head;
while (p != NULL) {
  printf("p= %p [Data= %d Next= %p]\n", p, p->Data, p->Next);
  p = p->Next;
}
```

Allocating list elements dynamically (5)

```
node *p, *q, *head;
int i;


printf("----- 5th try ---- \n");

p = (node *) malloc( sizeof(node) );
p->Data = 111;
head = p;   // first node
q = p;

p = (node *) malloc( sizeof(node) );
p->Data = 222;
q->Next = p;
q = p;

p = (node *) malloc( sizeof(node) );
p->Data = 333;
q->Next = p;
q = p;

p = (node *) malloc( sizeof(node) );
p->Data = 444;
q->Next = p;
p->Next = NULL;  // last node

p = head;
while (p != NULL) {
  printf("p= %p [Data= %d Next= %p] \n", p, p->Data, p->Next);
  p = p->Next;
}
```



q->Next = p

Allocating list elements dynamically (6)

```
  printf("----- 6th try ---- \n");


  for (i=0; i<4; ++i) {
      p = (node *) malloc( sizeof(node) );
      p->Data = 222;
      if (i==0) head = p; else q->Next = p;
      if (i==3) p->Next= NULL; else q = p;
  }

  p = head;
  while (p != NULL) {
    printf("p= %p [Data= %d Next= %p] \n", p, p->Data, p->Next);
    p = p->Next;
  }

  return 0;
}
```

References

[1] http://en.wikipedia.org/
[2]