CS101 Computer Programming and Utilization

Milind Sohoni

May 15, 2006

1. So far

2. Functions-PCAL implementation

3. call by value

4. call by reference

The story so far ...

- We have written some non-trivial programs
- We have seen various control flows.
- We have seen multi-dimensional arrays and the `char` data type.
- We saw how to get formatted output.
- We saw the use of functions

More Functions

We see in this talk (i) how functions are implemented, (ii) and certain calling methods. Finally, we solve some more non-trivial problems. Again www.cplusplus.com/doc/tutorial for reference.

How are functions implemented?

Consider the following simple C++ code:

```
#include <iostream.h>
int by2(int a)
{
   return(a/2);
}
int main()
{
  int N,x,y;
  cout << "N?";
  cin >> N;
  x=by2(N);
  y=by2(x);
  xout << y;
}
```

What issues arise in the translation of C++ intp PCAL?

- What is the translation of a function into PCAL?
- How is the argument/parameter to be passed to the function?
- How is the output to be received?
- How is the control flow to be implemented?

How are functions implemented?

Consider the following simple C++ code:

```
#include <iostream.h>
int by2(int a)
{
   return(a/2);
}
int main()
{
  int N,x,y;
  cout << "N?";
  cin >> N;
  x=by2(N);
  y=by2(x);
  xout << y;
}
```

What issues arise in the translation of C++ intp PCAL?

- What is the translation of a function into PCAL?
- How is the argument/parameter to be passed to the function?
- How is the output to be received?
- How is the control flow to be implemented?
- Allot different memory segments for the function and the amin program.

a	output	N	x	y
M10	M11	M1	M2	M3

How are functions implemented?

Consider the following simple C++ code:

```
#include <iostream.h>
int by2(int a)
{
   return(a/2);
}
int main()
{
  int N,x,y;
  cout << "N?";
  cin >> N;
  x=by2(N);
  y=by2(x);
  xout << y;
}
```

- Allot different memory segments for the function and the amin program.

a	output	N	x	y
M10	M11	M1	M2	M3

- Translate the function:

```
150 RCL M10;
    M11=M10 DIV 2;
    JUMP 25
```

And the main program

```
23 M10=M1 %copy N into input
24 JUMP 150
25 M2=M11 % copy output into
```

Call by Value

Consider the following simple C++ code:

```
#include <iostream.h>
int by2(int a)
{
   return(a/2);
}
int main()
{
  int N,x,y;
  cout << "N?";
  cin >> N;
  x=by2(N);
  y=by2(x);
  xout << y;
}
```

In other words,

- There is a separation of memories.
- The contents (values) of the input arguments are copied out into appropriate registers of the function.
- The function works out the answer.
- The output is copied back into appropriate registers in the calling program.
- Execution resumes.

This procedure is called CALL BY VALUE.

Call by Reference

Consider the following simple C++ code:

```
#include <iostream.h>
int by2(int a)
{
   return(a/2);
}
int main()
{
  int N,x,y;
  cout << "N?";
  cin >> N;
  x=by2(N);
  y=by2(x);
  xout << y;
}
```

There is another possible scenario:

- Create the function body as before.

```
RCL M10;
M11=M10 DIV 2
```

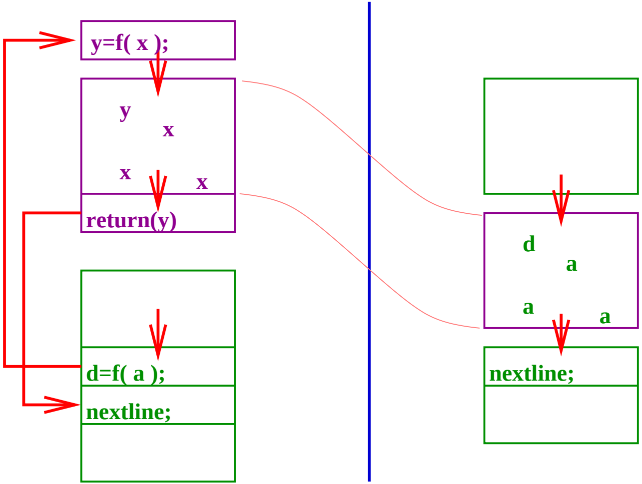
- For every function call, insert the function code in the main program, suitably modified:

```
RCL M1
M2=M1 DIV 2
```

Thus, the program code of the function is copied out into the main body and actually acts on the variables of the main program.
This is called Call by Reference.

Call by Value

Call by Ref.



```
#include <iostream.h>
int by2ref(int& a)
{
   b=a/2;
   a=a-2;
   return(b);
}
int by2value(int a)
{
   b=a/2;
   a=a-2;
   return(b);
}
int main()
{
  N=10;
  o1=by2value(N);
  o2=by2ref(N);
  o3=by2value(N);
}
```

The observed outputs will be:

o1	o2	o3
5	5	4

This is because:

- The first call by value by2val copied out N into its own space and returned the value 5.
- The second call by reference by2ref used the memory location N in its working and changed it to 8.
- The third call will now reflect $N/2 = 4$.

```
#include <iostream.h>
int by2ref(int& a)
{
   b=a/2;
   a=a-2;
   return(b);
}
int by2value(int a)
{
   b=a/2;
   a=a-2;
   return(b);
}
int main()
{
  N=10;
  o1=by2value(N);
  o2=by2ref(N);
  o3=by2value(N);
}
```

How do I specify Call by Reference?:

- Put an "&" after the type declaration that you want passed by reference. A function may have some arguments by value and others by reference.
- The calling syntax remains the same. Everything else remains the same.
- The caller does not know, without looking at the function definition, if his input parameters are going to change!.

Uses of Call by reference

- Having more than one outputs from a function.

The GCD problem

Recall that if g is the $\texttt{gcd}$ of m and n, then

$$g = \alpha m + \beta n$$

Write a program to compute g, α, β.

We use Euclid's algorithm.

Uses of Call by reference

- Having more than one outputs from a function.

The GCD problem

Recall that if g is the gcd of m and n, then

$$g = \alpha m + \beta n$$

Write a program to compute g, α, β.

We use Euclid's algorithm.

- If $m > n$ and $m = n \cdot q + r$, then

$$\text{gcd}(m, n) = \text{gcd}(n, r)$$

This is used to reduce the two arguments systematically.

Uses of Call by reference

- Having more than one outputs from a function.

The GCD problem

Recall that if g is the gcd of m and n, then

$$g = \alpha m + \beta n$$

Write a program to compute g, α, β.

We use Euclid's algorithm.

- If $m > n$ and $m = n \cdot q + r$, then

$$\text{gcd}(m, n) = \text{gcd}(n, r)$$

  This is used to reduce the two arguments systematically.
- At each step if m' and n' are such that
  - $\text{gcd}(m', n') = \text{gcd}(m, n)$.
  - Each m', n' is a linear combination of m, n.

Uses of Call by reference

- Having more than one outputs from a function.

The GCD problem

Recall that if g is the gcd of m and n, then

$$g = \alpha m + \beta n$$

Write a program to compute g, α, β.

We use Euclid's algorithm.

- If $m > n$ and $m = n \cdot q + r$, then

$$\gcd(m, n) = \gcd(n, r)$$

  This is used to reduce the two arguments systematically.
- At each step if m' and n' are such that
  - $\gcd(m', n') = \gcd(m, n)$.
  - Each m', n' is a linear combination of m, n.
- The above two steps are used recursively. If $m' = n' \cdot q' + r'$, then:
  - $\gcd(n', r') = \gcd(m', n') = \gcd(m, n)$.
  - Each n', r' is a linear combination of m, n.

Uses of Call by reference

- Having more than one outputs from a function.

```
#include <iostream.h>
#include <math.h>
void A(int a, int b,
          int& q, int& r)
{
   r=a%b;
   q=(a-r)/b;
   return;
}
```

Uses of Call by reference

- Having more than one outputs from a function.

```
#include <iostream.h>
#include <math.h>
void A(int a, int b,
          int& q, int& r)
{
   r=a%b;
   q=(a-r)/b;
   return;
}
```

We see here that `A(a,b,q,r)` have four arguments.

- The assumption is that $a > b$.
- a,b are the input arguments, passed by value.
- q,r are the output arguments, passed by reference.

The function implements:

$$a = b * q + r$$

Uses of Call by reference

Lets look at the main program:

- M,N are read in with $M > N$.
- m, n are the running arguments with the following invariants.
  - $m > n$.
  - $$\begin{array}{rcl} m & = & x[0]*m + x[1]*n \\ n & = & y[0]*m + y[1]*n \end{array}$$
- The next pair is $(m, n) \rightarrow (n, r)$, where

$$\begin{array}{rcl} r & = & m - q*n \\ & = & (x[0] - q*y[0])*m \\ & & +(x[1] - q*y[1])*n \end{array}$$

```
int main()
{
  int ...,x[2],y[2],t[2];
  x[0]=1; x[1]=0; y[0]=0; y[1]=1
  cout << "M>N?\n";
  cin >> M >> N;
  m=M; n=N;
  A(m,n,q,r);
  while (r!=0)
  {
    m=n; n=r;
    t[0]=x[0]-q*y[0];
    t[1]=x[1]-q*y[1];
    for (int i=0;i<2;i=i+1)
    {  x[i]=y[i];
       y[i]=t[i];
    }
    A(m,n,q,r);
  }
  cout << ...
}
```

Uses of Call by reference

- Having more than one outputs from a function.

```
[sohoni@nsl-13 lectures]$ ./a.out
M>N?
99 87
gcd=3  alpha=-7  beta=8

[sohoni@nsl-13 lectures]$ ./a.out
M>N?
115 78
gcd=1  alpha=19  beta=-28
```

```
int main()
{
  int ...,x[2],y[2],t[2];
  x[0]=1; x[1]=0; y[0]=0; y[1]=1
  cout << "M>N?\n";
  cin >> M >> N;
  m=M; n=N;
  A(m,n,q,r);
  while (r!=0)
  {
    m=n; n=r;
    t[0]=x[0]-q*y[0];
    t[1]=x[1]-q*y[1];
    for (int i=0;i<2;i=i+1)
    {  x[i]=y[i];
       y[i]=t[i];
    }
    A(m,n,q,r);
  }
  cout << ...
}
```

Uses of Call by reference

- Having more than one outputs from a function.
- Processing a large data-structure locally, without making copies.

Layer Fill

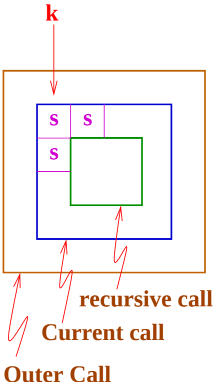
Fill up an $n \times N$ array in layers.

1	1	1
1	2	1
1	1	1

1	1	1	1
1	2	2	1
1	2	2	1
1	1	1	1

Strategy:

- Start with the outermost layer.
- Each call fills up the k-th layer and calls recursively, for the next layer.



```
void layer(int a[10][10],int k,
             int N, int start)
{ int low,hi,i,j;
  low=k; hi=N-k;
  if (low+1==hi){
     a[low][low]=start;
     return;
  }
  for (i=low;i<hi;i=i+1)
  {  for (j=low;j<hi;j=j+1)
     {if ((i==low) || (i==hi-1)
      || (j==low) || (j==hi-1))
          a[i][j]=start;
     };
  };
  if (low+1==hi-1) return;
  layer(a,k+1,N,start+1);
  return;
}
```

```
void layer(int a[10][10],int k,
              int N, int start)
{ int low,hi,i,j;
  low=k; hi=N-k;
  if (low+1==hi){
     a[low][low]=start;
     return;
  }
  for (i=low;i<hi;i=i+1)
  {  for (j=low;j<hi;j=j+1)
     {if ((i==low) || (i==hi-1)
      || (j==low) || (j==hi-1))
          a[i][j]=start;
     };
  };
  if (low+1==hi-1) return;
  layer(a,k+1,N,start+1);
  return;
}
```

Whats Happening

- The red code is the meat of the procedure.
- The green code is to terminate/continue the recursion.
- a is already filled correctly for 1,2,...,k-1.
- hi,low locate the boundaries.
- a is modified at the boundary and then a recursion.

```
void layer(int a[10][10],int k,
               int N, int start)
{ int low,hi,i,j;
  low=k; hi=N-k;
  if (low+1==hi){
     a[low][low]=start;
     return;
  }
  for (i=low;i<hi;i=i+1)
  {  for (j=low;j<hi;j=j+1)
     {if ((i==low) || (i==hi-1)
      || (j==low) || (j==hi-1))
          a[i][j]=start;
     };
  };
  if (low+1==hi-1) return;
  layer(a,k+1,N,start+1);
  return;
}
```

`layer.c`

- a: array always passed by reference, no need to declare it as such.
- k: layer to start
- N: array size
- start: the entry for layer k

```
int main()
{
  int a[10][10], N,i,j;
  cout << "N?\n";
  cin >> N;
  layer(a,0,N,1);
}
```

Assignments

- Write a program which on input N and k, outputs the $\binom{N}{k}$ subsets of $\{1, \ldots, N\}$ in an array of size $k \times \binom{N}{k}$. For example, for the input 4, 2 the following output is expected (upto column re-ordering):

1	1	1	2	2	3
2	3	4	3	4	4

- Let A be an $N \times N$ entries 0-1. Given $p = (i_0, j_0)$ and $p' = (i_1, j_1)$, we must check if there is a path in the matrix from p to p' which moves left/right/up/down, but does not visit any point (i, j) such that A[i][j]=0. See example below:

$(2,0) \longrightarrow (2,4)$

1	0	0	1	0	1
1	1	0	0	0	0
$\boxed{1}$	0	1	0	$\boxed{1}$	0
$\boxed{1}$	0	$\boxed{1}$	$\boxed{1}$	$\boxed{1}$	0
$\boxed{1}$	$\boxed{1}$	$\boxed{1}$	0	0	1
0	0	0	0	0	0