

Creating a Macintosh Maple Library

Suppose we wish to create a Maple Library called **mylib** on the **root hard drive** (usually named **Macintosh HD**) at **\mylib**. First create an empty folder called **mylib** on the **root hard drive**. Then execute the two following commands:

```
> restart;  
> march('create', "/mylib", 100);
```

The 100 is the maximum number of entries the library can hold. A library is only created once.

```
> restart;
```

This next command adds our library to the search tree for the libraries in use.

```
> libname:="/mylib", libname;
```

The next command checks to see which procedures are in the package **mypack** in the library and loads them into memory. This step should only be taken if procedures are already in the library.

```
> with(mypack);
```

We received the error message since the **mypack** package of procedures has yet to be created. Now we want to add Maple procedures called **bisection** and **bisection_dir** to **mypack**. To do this we first create the procedures.

```
> mypack[bisection] := proc(f::algebraic, a::numeric, b::numeric,  
    tol::positive, no::posint, root::name)  
    local A, B, TOL, C, OK, X, F, FA, FB, I, P, FP;  
    make sure endpoints and tolerance are of type floating point  
    A:=evalf(a);  
    B:=evalf(b);  
    TOL:=evalf(tol);  
    make sure A<B  
    if A > B then  
        X := A;  
        A := B;  
        B := X;  
    end if;  
    make f into a function F and evaluate at endpoints  
    F:=unapply(f, x);  
    FA:=F(A);  
    FB:=F(B);  
    check for errors in choice of endpoints  
    if A = B then  
        ERROR("The two endpoints of the interval [a,b] must be different");  
    else if FA*FB > 0 then ERROR("The function values at a and b must have opposite signs");  
    end if;  
    end if;  
    print table headings  
    printf(`  i      p                f(p)\n`);  
    printf(`  -      -                ----\n`);  
    execute the algorithm
```

STEP 1

```
I := 1;
```

STEP 2

```
OK := TRUE;  
while I <= no and OK = TRUE do
```

STEP 3

```

Compute  $p_i$ 
  C := (B - A) / 2.0;
  P := A + C;
STEP 4
  FP := F(P);
  printf(`%3d    %15.8e    %15.7e \n`, I, P, FP);
  if abs(FP) < 1.0e-20 or C < TOL then
procedure completed successfully
  printf(`\nThe approximate solution is %a`, args[6]);
  printf(` = %11.8f \n`, P);
  printf(`with f(%a`, args[6]);
  printf(`) = %12.8f\n`, FP);
  root:=P;
  OK:=FALSE;
  else
STEP 5
  I := I+1;
STEP 6
compute  $a_i$  and  $b_i$ 
  if FA*FP > 0 then
  A := P;
  FA := FP;
  else
  B := P;
  FB := FP;
  end if;
  end if;
  end do;
  if OK = TRUE then
STEP 7
procedure completed unsuccessfully
  printf(`\nIteration number %3d`, no);
  printf(` gave approximation %12.8f\n`, P);
  printf(` F(P) = %12.8f not within tolerance : %15.8e\n`, FP, TOL);
  RETURN();
  else
  P;
  end if;
  end proc;

> mypack[bisection_dir]:=proc()
  printf(`bisection returns a root of the given function.\n\n`);
  printf(`The arguments for bisection are:\n`);
  printf(`(1)function expression in x\n`);
  printf(`(2)left end point\n`);
  printf(`(3)right end point\n`);
  printf(`(4)tolerance\n`);
  printf(`(5)maximum number of iterations\n`);
  printf(`(6)variable for returning root.\n\n`);
  printf(`If assigning the result to a variable, have the\n`);
  printf(`variable and the 6th argument the same.\n\n`);
  printf(`If r is the variable for returning the root\n`);
  printf(`and has already been given a value,\n`);
  printf(`the procedure should be preceded by the statement:\n`);
  printf(`r:='r'`);
  end;

```

Next we save the procedures in the package **mypack** in the library **mylib**. The command [savlibname](#) tells Maple in which library to save the package containing our procedures.

```
> savelibname:="/mylib";
```

The command `savelib` is then used to save the package into the library at that location.

```
> savelib(mypack);
```

Now we check to see that the procedures are really there and load them.

```
> restart;
```

```
> libname:="/mylib",libname;
```

```
> with(mypack);
```

Notice that the two new procedures have been added to the library. Suppose we now want to add two new procedures, `chop` and `chop_dir`, to our library. We create these procedures.

```
> mypack[chop]:=proc(x::numeric,t::posint,answer::name)
  local e, x2;
  if x=0 then x2:=0;
  else
  e:=trunc(evalf(log10(abs(x))));
  if abs(x)>1 then e:=e+1 fi;
  x2:=evalf(trunc(x*10^(t-e))*10^(e-t));
  fi;
  printf(`\n%a`,args[3]);
  printf(` = %11.8f \n`,x2);
  answer:=x2;
  x2;
end;
```

```
> mypack[chop_dir]:=proc()
  printf(`chop reduces a number to the specified number of digits.
  \n\n`);
  printf(`The arguments for chop are:\n`);
  printf(`(1)an integer, fraction, or floating point number, (2)the
  number of digits to chop to, (3)variable for returning answer.\n`);
  ;
  printf(`If assigning the result to a variable, have the\n`);
  printf(`variable and the 3rd argument the same.\n\n`);
  printf(`If r is the variable for returning the answer and has
  already been given a value, the procedure should be preceded by
  the statement:\n`);
  printf(`r:='r'`);
end;
```

We add these to our library by proceeding as before.

```
> savelibname:="/mylib";
```

```
> savelib(mypack);
```

Now we check to see that all of the procedures are really there and load them.

```
> restart;
```

```
> libname:="/mylib",libname;
```

```
> with(mypack);
```

```
>
```

We now test our procedures.

```
> bisection_dir();
```

```
> r:='r';
```

```
> bisection(cos(x),0.5,3.0,.00000001,100,r);
```

```
> chop_dir();
```

```
> r:='r';
```

```
> r:=chop(-151/13,5,r);
```

```
> r;
```

```
>
```

