

```

PRIVATE runexperiment(int G)
/* run experiment for G generations */
{
    int i,k,sum;

    gen = 0; sum = 0;
    while( gen++ < G ) {

        for( i=k; i<POP; i++ ) {
            sethome();
            runaprog( pgm[i] );
            pfit[i] = evalfitness();
            success[i] = finishF;
        } /* end one generation */

        for( sum=0, i=0; i<POP; i++ )
            sum += success[i];
        keepbest();
        stat1(sum);

        if(gen == G) return;

        docrossover();
        for(i=0; i<100; i++)
            pgm[i+200] = addprog( copyprog(pgm[i]));
        for(i=0; i<100; i++)
            pgm[i+300] = extendprog( copyprog(pgm[i]));
    }
}

/* pick parents with replacement, parents must be diff */
PRIVATE docrossover()
{
    int k,ap1,ap2,r,s;
    for( k=40; k<200; k+=2 ) {
        r = myrnd(NBEST);
        do { s = myrnd(NBEST); } while( s == r );
        ap1 = copyprog(pgm[r]);
        ap2 = copyprog(pgm[s]);
        crossover(ap1,ap2);
        pgm[k] = ap1;
        pgm[k+1] = ap2;
    }
}

```

```

PRIVATE initprog()
{
    int i,k;
    for( i=0; i<POP; i++)
        pgm[i] = genprogn(PLEN,&k);
}

=====

PRIVATE int genprog1() /* random prog size 1 cell */
{
    int ap;

    ap = getacell();
    cmd(ap) = myrnd(3) + IFAND;
    arg1(ap) = myrnd(IFAND);
    arg2(ap) = myrnd(IFAND);
    arg3(ap) = myrnd(IFAND);
    if( is4arg(cmd(ap)) ) arg4(ap) = myrnd(IFAND);
    return ap;
}

PRIVATE int genprog(int depth)
/* generate a random program */
/* depth is used to track the level, start with n */
{
    int ap, c, i;

    if ( depth == 0 ) return myrnd(IFAND);
                                /* terminals only */
    c = myrnd(ENDCODE);
    if ( isterm(c) ) return c;
    ap = getacell();
    cmd(ap) = c;
    arg1(ap) = genprog(depth-1);
    arg2(ap) = genprog(depth-1);
    arg3(ap) = genprog(depth-1);
    if( is4arg(cmd(ap)) )
        arg4(ap) = genprog(depth-1);
    return ap;
}

```

```

int proglen(int ap)
/* find length of a prog, (the number of symbol) */
{
    int len;

    if( isterm(ap) ) return 1;
    len = 1 + proglen(arg1(ap)) +
          proglen(arg2(ap)) + proglen(arg3(ap));
    if ( is4arg(cmd(ap)) )
        len += proglen(arg4(ap));
    return len;
}

int genprogn(int n, int *len)
/* gen random prog length > n, return length in len */
{
    int ap, k;

    k = n / 10 + 2;
    /* depth limit : n < 10 = 2, n < 20 = 3 etc. */
    while ( 1 ) {
        ap = genprog(k);
        *len = proglen(ap);
        if (*len > n) break;
    }
    return ap;
}

tofile(int ap, FILE *fp)
/* output a compact form of a program */
{
    if ( ap == NIL ) return;
    if (isterm(ap)) {
        fputc(ap+'a',fp);
        return;
    }
    fputc(cmd(ap)+'a',fp);
    tofile(arg1(ap),fp);
    tofile(arg2(ap),fp);
    tofile(arg3(ap),fp);
    if( is4arg(cmd(ap)) ) tofile(arg4(ap),fp);
}

```

```

int fromfile(FILE *fp)
/* input a compact form of a program */
{
    int c,ap;

    c = fgetc(fp);
    if ( c == EOF || c < 'a' || c > 'n' ) return NIL;
    c -= 'a';
    if ( isterm(c) ) return c;
    ap = getacell();
    cmd(ap) = c;
    arg1(ap) = fromfile(fp);
    arg2(ap) = fromfile(fp);
    arg3(ap) = fromfile(fp);
    if( is4arg(cmd(ap)) ) arg4(ap) = fromfile(fp);
    return ap;
}

/* ----- genetic operations ----- */

#define tosscoin() (myrnd(3) == 0)
/* is true with prob 1/3 */

int copyprog(int ap)
/* copy program tree, return pointer to new tree */
{
    int ap2;

    if ( isterm(ap) ) return ap;
    ap2 = getacell();
    cmd(ap2) = cmd(ap);
    arg1(ap2) = copyprog(arg1(ap));
    arg2(ap2) = copyprog(arg2(ap));
    arg3(ap2) = copyprog(arg3(ap));
    if( is4arg(cmd(ap)) ) arg4(ap2) = copyprog(arg4(ap));
    return ap2;
}

```

```

/* find pointer to a n th compenent of ap, return yes/no
   and ptr , ptr is undef if not find
*/

```

```

PRIVATE int findnode(int ap, int *n, int *ptr)
{
    int i,k,c;

    if ( *n == 0 ) { *ptr = ap; return 1; }
    c = cell[ap];
    if ( isterm(c) ) { (*n)--; return 0;}
    if ( c < ENDCODE ) {
        (*n)--;
        k = argsize(c);
        for( i=1; i<=k; i++ )
            if ( findnode( ap+i, n, ptr) ) return 1;
        return 0;
    }
    return findnode( c, n, ptr);
}

```

```

crossover(int ap1, int ap2)
/* swap two pgms at two random positions */
{
    int p1,p2,a,b,temp;

    p1 = myrnd(proglen(ap1)-1) + 1;
    p2 = myrnd(proglen(ap2)-1) + 1;
    a = ap1; b = ap2;
    findnode(ap1,&p1, &a);
    findnode(ap2,&p2, &b);
    temp = cell[a];
    cell[a] = cell[b];
    cell[b] = temp;
}

```

```

int addprog(int ap) /* add a random prog at root */
{
    int ap2,i,a;
    ap2 = genprogl();
    i = myrnd(proglen(ap2) -1) +1;
    a = ap;
    if( findnode(ap2, &i, &a) ) cell[a] = ap;
    return ap2;
}

```

```

}

int extendprog(int ap)
/* extend a terminal node with a random prog */
{
    int ap2,k,i,a,s;

    ap2 = genprog1();
    k = proglen(ap) -1;
    do {
        i = myrnd(k)+1;
        a = ap;
        findnode(ap, &i, &a);
    } while ( ! (s = isterm(cell[a])) );
    if( s ) cell[a] = ap2;
    return ap;
}

int mutateprog(int ap)
/* create a mutant with probab according to tosscoin() */
{
    int ap2, b, c;

    if( isterm(ap) )
        return (tosscoin() ? myrnd(IFAND) : ap);
    ap2 = getacell();
    c = tosscoin() ? myrnd(3) + IFAND : cmd(ap);
    cmd(ap2) = c;
    arg1(ap2) = mutateprog(arg1(ap));
    arg2(ap2) = mutateprog(arg2(ap));
    arg3(ap2) = mutateprog(arg3(ap));
    if( is4arg(c) )
        arg4(ap2) = is4arg(cmd(ap)) ? mutateprog(arg4(ap)) :
            genprogn(2,&b);
    return ap2;
}

```

=====

```

int evalprog(int ap)
/* evaluate a prog once, affect ds,de,dw and flags */
{
    int s1,s2;

    if( isterm(ap) ) {
        switch( ap ) {
            case SP : ds++; return 1;
            case SM : ds--; return 1;
            case EP : de++; return 1;
            case EM : de--; return 1;
            case WP : dw++; return 1;
            case WM : dw--; return 1;
            case HITQ : return hitF;
            case SEEQ : return seeF;
            case INCQ : return incF;
            case DECQ : return decF;
            case OUTQ : return outF;
        }
        return 0;
    }
    switch( cmd(ap) ) {
        case IFAND :
            s1 = evalprog(arg1(ap));
            s2 = evalprog(arg2(ap));
            if( s1 && s2 ) return evalprog(arg3(ap));
            else return evalprog(arg4(ap));
        case IFOR :
            s1 = evalprog(arg1(ap));
            s2 = evalprog(arg2(ap));
            if( s1 || s2 ) return evalprog(arg3(ap));
            else return evalprog(arg4(ap));
        case IFNOT :
            if( ! evalprog(arg1(ap)) ) return evalprog(arg2(ap));
            else return evalprog(arg3(ap));
    }
    return 0;
}

```