**Описание функции Аккермана**

На сегодняшний день функция Аккермана является одной из самых трудно вычислимых функций, так как она дважды рекурсивна, т.е. она не только определяется через себя, но и является одним из параметров вызываемой функции.

функция Аккермана (А) от трех параметров, индуктивно задается на трех неотрицательных целых числах и выглядит следующим образом :

| X+1, если N=0

| X, если N=1, Y=0,

| 0, если N=2, Y=0,

A(N,X,Y)= | 1, если N=3, Y=0,

| 2, если N=>4, Y=0,

| A(N-1,A(N,X,Y-1),X), если N#0, Y#0;

где N,X,Y - целые неотрицательные числа

* функция Аккермана (А) от двух параметров, индуктивно задается на паре неотрицательных целых числах:

Значение этой функции очень быстро возрастает по мере увеличения первого из ее параметров(“ m”). Рассмотрим функцию Аккермана первого порядка A(1, n): зафиксируем первый параметр и будем постепенно увеличивать второй.

A(1,0)=A(0,1)=1+1=2

A(1,1)=A(0,A(1,0))=1+A(1,0)=1+2=3

A(1,2)=A(0,A(1,1))=1+A(1,1)=1+1+A(1,0)=2+2=4

A(1,3)= A(0, A(1,2))=1+ A(1,2)=1+1+ A(1,1)=1+1+1+ A(1,0)=3+2=5

…

A (1, n )= n +2

Наблюдается линейная зависимость между значением функции и параметром “ n”. Аналогично рассмотрим функцию Аккермана второго порядка:

A(2,0)= A(1,1)=1+2=3

A(2,1)= A(1, A(2,0))= A(0, A(1,( A(2,0)-1)))=2+3=5

A(2,2)=A(1,A(2,1))=A(1,5)=A(0,A(1,4))=A(0,A(0,A(1,3)))=…=1+1+1+1+3=4+3=7

…

A(2,n)=2\*n+3

Таким образом, были получены формулы для вычисления функции Аккермана со степенью m=2,3,4.

Анализируя полученные выражения, делаем вывод, что функция Аккермана уже третей степени обладает степенной зависимостью от параметра « m», а для четвертой степени – характер зависимости многостепенной.

**Разработка алгоритма программы**

Для построения алгоритма программы используем сведения, изложенные в разделе 1 о рекурсивных функциях, а также описание свойств функции Аккермана из предыдущего параграфа.

Необходимо разработать программу, вычисляющую рекурсивную функцию, т.е. из функции main вызывается функция Akk:

main() /\* вызывающая функция \*/

{... Akk()...}

Akk() /\* рекурсивная функция \*/

{ ... Akk()...}

Рекурсивная функция Akk имеет параметры Р1,Р2,...,Рs, внутренние переменные V1,V2,...,Vt и в функциях main и Akk имеется k обращений к функции Akk. Для реализации такой функции требуются следующие дополнительные объекты:

* переменные AR1,AR2,...,ARs, содержащие значения фактических параметров при вызове функции Akk (типы переменных должны соответствовать типам параметров Р1,Р2,...,Рs);
* переменная rz для вычисляемого функцией Akk результата (тип переменных совпадает с типом возвращаемого значения функции Akk);
* стек, содержащий в себе все параметры и все внутренние переменные функции Akk, а также переменную lr типа int, для хранения точки возврата, и переменную pst типа указатель, для хранения адреса предыдущего элемента стека;
* указатель dl для хранения адреса вершин стека;
* промежуточный указатель u для операций над стеком;
* k новых меток L1,...,Lk для обозначенных точек возврата;
* промежуточная переменная l типа int для передачи номера точки возврата.

Очевидно, что помимо непосредственного вычисления функции Аккермана, программа должна содержать некоторый подготовительный этап и завершающий этап. На подготовительном этапе необходимо реализовать считывание параметров функции, а также проверку их принадлежности множеству допустимых данных функции. Завершающий этап должен содержать функцию вывода и сохранения полученных результатов.

В результате получим следующий набор действий, выполняемый программой:

1) считывание исходных данных;

2) выбор теста;

3) выполнение выбранного теста;

4) вывод результатов тестирования на экран;

5) запись результатов в файл

Основная логика программы, реализующая вычисление функции Аккермана, будет включена в функцию Akkerman.

В алгоритм введено понятие «объект» и анализ полей «объекта». Объект представляет собой структуру со следующими полями:

* Am – хранит первый параметр функции Аккермана
* An – хранит второй параметр функции Аккермана
* Rz – в эту переменную записывается результат вычислению данной функции
* L – уровень текущей функции, подробно это поле будет описано ниже
* Указатель на такую же структуру – это поле необходимо для включения вновь созданного объекта в стек

Таким образом, каждый объект представляет собой функцию Аккермана со своими параметрами. При рекурсивном вызове функции, создается новый объект и помещается в начало стека. Каждому, вновь созданному, объекту присваивается определенный уровень “ L”: L=1 – означает, что данная функция Аккермана последняя в стеке и вычисление закончено; L=2 – означает, что данная функция порождает новую функцию Аккермана со следующими параметрами: А( am, an-1); L=3 – означает, что надо вызывать новую функцию Аккермана с параметрами A( am-1, rz), где rz – это результат вычисления функции Аккермана на предыдущем шаге.

Отметим, что определение функции Аккермана вполне можно назвать рекурсивным одно значение этой функции определяется через другие, с меньшим первым аргументом. Оно является примером рекурсивного определения, не сводящегося к примитивной рекурсии.

Следующая программа вычисляет функцию Аккермана с использованием рекурсивной функции ackr и вспомогательной функции smacc:   
  
  
  /\* рекурсивное вычисление функции Аккермана \*/  
  # include   
  main () /\* вызывающая \*/  
  { int x,y,n,t;  /\* функция \*/  
  int ackr(int, int, int);  
  scanf("%d %d %d",&n,&x,&y);  
  t=ackr(n,x,y);  
  printf("%d",t);  
  }  
  int smacc( int n,int x ) /\* вспомогательная \*/  
  { switch (n) /\* функция \*/  
  { case 0: return(x+1);  
  case 1: return (x);  
  case 2: return (0);  
  case 3: return (1);  
  default: return (2);  
  }  
  }  
  int ackr( int n, int x, int y) /\* рекурсивная \*/  
  { int z;  /\* функция \*/  
  int smacc( int,int);  
  if(n==0 || y==0) z=smacc(n,x);  
  else { z=ackr(n,x,y-1);  /\* рекурсивные \*/  
  z=ackr(n-1,z,x);  } /\* вызовы ackr(...) \*/  
  return z;  
  }  
  
  
  
Модифицируя функции main и ackr в соответствии с изложенным методом получим следующую программу:   
  
  
  /\* Эквивалентная нерекурсивная программа \*/  
  /\* для вычисления функции Аккермана \*/  
  #include   
  #include   
  int main()  
  { typedef struct st  
  { int i,j,k,z,lr;  
  struct st \*pst;  
  } ST;  
  ST \*u, \*dl=NULL;  
  int l,x,y,n;  
  int smacc(int,int);  
  int an,ax,ay,rz,t;  
  scanf("%i %i %i",&n,&x,&y);  
  an=n;ax=x;ay=y;l=1;  /\* - замена вызова - \*/  
  goto ackr;  /\* t=ackr(n,x,y);  \*/  
  l1: t=rz;  /\* - - - - - - - - \*/  
  printf("\n %d ",t);  
  goto jackr;  
  /\* начало фрагмента заменяющего функцию ackr \*/  
  ackr:  
  u=( ST \*) malloc( sizeof (ST) );  
  u->i=an;  
  u->j=ax;  
  u->k=ay;  
  u->lr=l;  
  u->pst=dl;  
  dl=u;  
  if (an==0||ay==0)  
  dl->z=smacc(an,ax);  
  else  
  {  
  an=dl->i;  /\* - замена вызова - \*/  
  ax=dl->j;  /\* \*/  
  ay=dl->k-1;  /\* z=ackr(n,x,y-1);  \*/  
  l=2;  /\* \*/  
  goto ackr;  /\* \*/  
  l2: dl->z=rz;  /\* - - - - - - - - \*/  
  an=dl->i-1;  /\* - замена вызова - \*/  
  ax=rz;  /\* \*/  
  ay=dl->j;  /\* z=ackr(n-1,z,x);  \*/  
  l=3;  /\* \*/  
  goto ackr;  /\* \*/  
  l3: dl->z=rz;  /\* - - - - - - - - \*/  
  }  
  rz=dl->z;  /\* - - - - - - - - \*/  
  an=dl->i;  /\* \*/  
  ax=dl->j;  /\* замена \*/  
  ay=dl->k;  /\* \*/  
  l=dl->lr;  /\* оператора \*/  
  u=dl;  /\* \*/  
  dl=u->pst;  /\* return z ;  \*/  
  free(u);  /\* \*/  
  switch(l) /\* \*/  
  { case 1: goto l1;  /\* \*/  
  case 2: goto l2;  /\* \*/  
  case 3: goto l3;  /\* \*/  
  } /\* - - - - - - - - \*/  
  jackr:  
  }  
  int smacc( int n,int x ) /\* вспомогательная функция \*/  
  { switch (n)  
  { case 0: return(x+1);  
  case 1: return (x);  
  case 2: return (0);  
  case 3: return (1);  
  default: return (2);  
  }  
  }

Следующая программа вычисляет функцию Аккермана с использованием рекурсивной функции ackr и вспомогательной функции smacc:   
  
      /\*      рекурсивное  вычисление функции Аккермана    \*/  
      # include   
      main ()                          /\*  вызывающая    \*/  
      {  int x,y,n,t;                  /\*  функция        \*/  
          int ackr(int, int, int);  
          scanf("%d %d %d",&n,&x,&y);  
          t=ackr(n,x,y);  
          printf("%d",t);  
      }  
      int smacc( int n,int x )          /\*  вспомогательная  \*/  
      {  switch (n)                    /\*  функция        \*/  
          {  case 0:  return(x+1);  
              case 1:  return (x);  
              case 2:  return (0);  
              case 3:  return (1);  
              default: return (2);  
            }  
      }  
      int ackr( int n, int x, int y)    /\*  рекурсивная      \*/  
      {  int z;                        /\*  функция          \*/  
        int smacc( int,int);  
        if(n==0 || y==0)  z=smacc(n,x);  
        else { z=ackr(n,x,y-1);        /\*  рекурсивные      \*/  
                z=ackr(n-1,z,x);  }    /\*  вызовы ackr(...) \*/  
        return z;  
      }  
  
Модифицируя функции main и ackr в соответствии с изложенным методом получим следующую программу:   
  
      /\*      Эквивалентная нерекурсивная программа      \*/  
      /\*      для вычисления функции Аккермана            \*/  
    #include   
    #include   
    int main()  
    {  typedef struct st  
        { int i,j,k,z,lr;  
          struct st \*pst;  
        } ST;  
        ST \*u, \*dl=NULL;  
        int l,x,y,n;  
        int  smacc(int,int);  
        int an,ax,ay,rz,t;  
        scanf("%i %i %i",&n,&x,&y);  
        an=n;ax=x;ay=y;l=1;      /\*  -  замена вызова    -  \*/  
        goto ackr;                /\*    t=ackr(n,x,y);      \*/  
    l1: t=rz;                    /\*  -  -  -  -  -  -  -  -  \*/  
        printf("\n %d ",t);  
        goto jackr;  
        /\*    начало фрагмента заменяющего функцию  ackr      \*/  
    ackr:  
        u=( ST \*) malloc( sizeof (ST) );  
        u->i=an;  
        u->j=ax;  
        u->k=ay;  
        u->lr=l;  
        u->pst=dl;  
        dl=u;  
        if (an==0||ay==0)  
        dl->z=smacc(an,ax);  
        else  
            {  
                an=dl->i;        /\*  -  замена вызова    -  \*/  
                ax=dl->j;        /\*                          \*/  
                ay=dl->k-1;      /\*    z=ackr(n,x,y-1);    \*/  
                l=2;              /\*                          \*/  
                goto ackr;        /\*                          \*/  
          l2:  dl->z=rz;        /\*  -  -  -  -  -  -  -  -  \*/  
                an=dl->i-1;      /\*  -  замена вызова    -  \*/  
                ax=rz;            /\*                          \*/  
                ay=dl->j;        /\*    z=ackr(n-1,z,x);    \*/  
                l=3;              /\*                          \*/  
                goto ackr;        /\*                          \*/  
          l3:  dl->z=rz;        /\*  -  -  -  -  -  -  -  -  \*/  
            }  
        rz=dl->z;                /\*  -  -  -  -  -  -  -  -  \*/  
        an=dl->i;                /\*                          \*/  
        ax=dl->j;                /\*      замена            \*/  
        ay=dl->k;                /\*                          \*/  
        l=dl->lr;                /\*      оператора          \*/  
        u=dl;                    /\*                          \*/  
        dl=u->pst;                /\*      return z ;        \*/  
        free(u);                  /\*                          \*/  
        switch(l)                /\*                          \*/  
            {  case 1: goto l1;  /\*                          \*/  
                case 2: goto l2;  /\*                          \*/  
                case 3: goto l3;  /\*                          \*/  
            }                    /\*  -  -  -  -  -  -  -  -  \*/  
    jackr:  
    }  
    int smacc( int n,int x )      /\* вспомогательная функция \*/  
    {  switch (n)  
            { case 0:  return(x+1);  
              case 1:  return (x);  
              case 2:  return (0);  
              case 3:  return (1);  
              default: return (2);  
            }  
    }

Вставить модифицированное тело функции f в конце функции main.

Для иллюстрации изложенного рассмотрим несколько вариантов реализации программы вычисляющей функцию Аккермана, которая определяется так:

+ X+1, если N=0

| X, если N=1, Y=0,

| 0, если N=2, Y=0,

A(N,X,Y)= | 1, если N=3, Y=0,

| 2, если N=>4, Y=0,

+ A(N-1,A(N,X,Y-1),X), если N#0, Y#0;

где N,X,Y - целые неотрицательные числа.

Следующая программа вычисляет функцию Аккермана с использованием рекурсивной функции ackr и вспомогательной функции smacc:

/\* рекурсивное вычисление функции Аккермана \*/

# include

main () /\* вызывающая \*/

{ int x,y,n,t; /\* функция \*/

int ackr(int, int, int);

scanf("%d %d %d",&n,&x,&y);

t=ackr(n,x,y);

printf("%d",t);

}

int smacc( int n,int x ) /\* вспомогательная \*/

{ switch (n) /\* функция \*/

{ case 0: return(x+1);

case 1: return (x);

case 2: return (0);

case 3: return (1);

default: return (2);

}

}

int ackr( int n, int x, int y) /\* рекурсивная \*/

{ int z; /\* функция \*/

int smacc( int,int);

if(n==0 || y==0) z=smacc(n,x);

else { z=ackr(n,x,y-1); /\* рекурсивные \*/

z=ackr(n-1,z,x); } /\* вызовы ackr(...) \*/

return z;

}

Модифицируя функции main и ackr в соответствии с изложенным методом получим следующую программу:

/\* Эквивалентная нерекурсивная программа \*/

/\* для вычисления функции Аккермана \*/

#include

#include

int main()

{ typedef struct st

{ int i,j,k,z,lr;

struct st \*pst;

} ST;

ST \*u, \*dl=NULL;

int l,x,y,n;

int smacc(int,int);

int an,ax,ay,rz,t;

scanf("%i %i %i",&n,&x,&y);

an=n;ax=x;ay=y;l=1; /\* - замена вызова - \*/

goto ackr; /\* t=ackr(n,x,y); \*/

l1: t=rz; /\* - - - - - - - - \*/

printf("\n %d ",t);

goto jackr;

/\* начало фрагмента заменяющего функцию ackr \*/

ackr:

u=( ST \*) malloc( sizeof (ST) );

u->i=an;

u->j=ax;

u->k=ay;

u->lr=l;

u->pst=dl;

dl=u;

if (an==0||ay==0)

dl->z=smacc(an,ax);

else

{

an=dl->i; /\* - замена вызова - \*/

ax=dl->j; /\* \*/

ay=dl->k-1; /\* z=ackr(n,x,y-1); \*/

l=2; /\* \*/

goto ackr; /\* \*/

l2: dl->z=rz; /\* - - - - - - - - \*/

an=dl->i-1; /\* - замена вызова - \*/

ax=rz; /\* \*/

ay=dl->j; /\* z=ackr(n-1,z,x); \*/

l=3; /\* \*/

goto ackr; /\* \*/

l3: dl->z=rz; /\* - - - - - - - - \*/

}

rz=dl->z; /\* - - - - - - - - \*/

an=dl->i; /\* \*/

ax=dl->j; /\* замена \*/

ay=dl->k; /\* \*/

l=dl->lr; /\* оператора \*/

u=dl; /\* \*/

dl=u->pst; /\* return z ; \*/

free(u); /\* \*/

switch(l) /\* \*/

{ case 1: goto l1; /\* \*/

case 2: goto l2; /\* \*/

case 3: goto l3; /\* \*/

} /\* - - - - - - - - \*/

jackr:

}

int smacc( int n,int x ) /\* вспомогательная функция \*/

{ switch (n)

{ case 0: return(x+1);

case 1: return (x);

case 2: return (0);

case 3: return (1);

default: return (2);

}

}