ELEKTRIJADA 2016

Rimini (Italy), May 12-17, 2016

INFORMATICS

Tasks

❶ **Find the output of the following program:**

```
#include <stdio.h>
#define M1(X) printf("%d",printf("""X"""))
#define M2(X) printf("%d",printf("""X\n"""))
#define M3(X,O,Y) X O Y
int main()
{
   printf("1:%d\n", M1(999)||M1(88));
   printf("2:%d\n", M3(M1(777),||,M1(66)));
   printf("3:%d\n", M3(M2(555),&&,M2(44)));
   return 0;
}
```

❷ **Find the output of the following program:**

```
#include <stdio.h>
#include <stdlib.h>
#define QN (NODE*)calloc(1,sizeof(NODE))
typedef struct _n
   { char c; int i; struct _n *r,*d; } NODE;
char cc, nc;
void f1(NODE *q, int i)
{
  if (q)
  {
    if (q->i==i)
       printf("%c:%d\n", cc=q->c, q->i);
    if (q->c==cc) nc++;
    if (q->d) f1(q->d,i);
    f1(q->r,i);
  }
}
int fm(NODE *q)
{
  int m=-1, m1=-1, m2=-1;
  if (q)
  {
    if (q->i>m) m=q->i;
    if (q->d) m1=fm(q->d), m=(m1>m)?m1:m;
    m2=fm(q->r), m=(m2>m)?m2:m;
  }
  return m;
}
void add(NODE **p, char *s)
{
  if (*s) if (*p==NULL) *p=QN,(*p)->c=*s;
    else if ((*p)->c<*s) add(&((*p)->r),s);
    else if ((*p)->c>*s) add(&((*p)->d),s);
    else (*p)->i++, add(&((*p)->r),s);
}
void main()
{
  NODE *f=NULL; char *w="ELEKTRIJADA RIMINI";
  for (char *t=w; *t; add(&f,t++));
  f1(f,fm(f)); printf("%d",nc);
}
```

❸ **Find the output of the following program:**

```
#include <stdio.h>
int main()
{
   unsigned int i,x,y,z,w=3;
   for (i=10; i; i--)
   {
     for (y=0; ~i<<y>>(sizeof(y)*8-1); y++);
     y=sizeof(y)*8-y;
     for (y--, z=0, x=i; x>>y&1; y--, z++);
     if (z==w) printf("%d:%d\n", z, i), w--;
   }
   return 0;
}
```

❹ **Find the output of the following program:**

```
#include <stdio.h>
int x=-1, y, i, bl=2;
void fh(int d, int r, int o)
{
   if (!o) return;
   d+=r; fh(d,-r,o-1); step(d);
   d-=r; fh(d,r,o-1); step(d);
   fh(d,r,o-1); d-=r; step(d);
   fh(d,-r,o-1);
}
void b(unsigned k, int kl, char *s)
{
   s[kl]=0;
   for (int i=kl-1; i>=0; k>>=1, i--)
      s[i]=k&1|0x30;
}
void step(int dir)
{
   char xx[3], yy[3];
   switch(dir & 3)
   {
      case 0: x++; break;
      case 1: y++; break;
      case 2: x--; break;
      case 3: y--; break;
   }
   b(x,bl,xx);
   b(y,bl,yy);
   if (i%5==0)
      printf("%d:%s-%s\n", i/5+1, xx, yy);
   i++;
}
int main()
{
  step(0);
  fh(0,1,bl);
  return 0;
}
```

❺ **Find the output of the following program:**

```
#include <stdio.h>
#include <stdlib.h>
typedef struct _n
  { char c; int n; struct _n **p; } NODE;
typedef struct _in
  { char c; struct _in *p } IN;
NODE *t,*h; IN in[100]; int nt;
NODE *newNode(char c)
{
  NODE *q=(NODE*)calloc(1,sizeof(NODE));
  q->c=c; return q;
}
NODE *fs(IN q[], char c)
{
  for (int i=0; i<nt; i++)
    if (q[i].c==c) return q[i].p;
  return NULL;
}
void fc(char *s)
{
  int i=0, j=0, k=0;
  if (!h)
  {
    t=h=newNode(*s);
    in[nt].c=*s; in[nt].p=t; nt++; s++;
  }
  for(;*s;s++)
  {
    NODE **tx=
       (NODE**)calloc(t->n+1,sizeof(NODE*));
    for (i=0; i<t->n; i++) *(tx+i)=*(t->p+i);
    if (!(*(tx+i)=fs(in,*s)))
    {
      *(tx+i)=newNode(*s);
      in[nt].c=*s; in[nt].p=*(tx+i); nt++;
    }
    if (t->p) free(t->p);
    ++t->n; t->p=tx; t=t->p[t->n-1];
  }
}
void ft(NODE *q)
{
  static char s[100]; static int si;
  if (q)
  {
    s[si++]=q->c;
    for (int j=0; j<strlen(s)-1; j++)
      if (s[j]==q->c) printf("%s\n", &s[j]);
    if (q->n>0)
    {
      NODE *n=*q->p, **tx=NULL;
      if (q->n>1)
      {
        tx = (NODE**)
           calloc(q->n-1,sizeof(NODE*));
        for (int i=1; i<q->n; i++)
          *(tx+i-1)=*(q->p+i);
      }
      free(q->p);
      q->n=q->n-1;q->p=tx; ft(n);
    }
  }
}
void main() { fc("RIMINI"); ft(h); }
```

❻ **Find the output of the following program:**

```
#include <stdio.h>
#define ML(x,y) (strlen(x)+strlen(y))
#define MR(x,y) (x!=y)
#define MX(x,y) x+=y, y=x-y, x-=y
int main()
{
  unsigned char f[]="%*2s%03s";
  char *t="ELEKTRIJADA 2016 RIMINI";
  char a[20]="", b[20]="";
  for( ; sscanf(t,f,a,b); t+=ML(a,b))
  {
     for(int j=0; j<strlen(f)/2; j++)
       if MR(*(f+j),*(f+j+strlen(f)/2))
          MX((*(f+j)),(*(f+j+strlen(f)/2)));
     if (strlen(a)==2) printf("%s\n", a);
  }
  return 0;
}
```

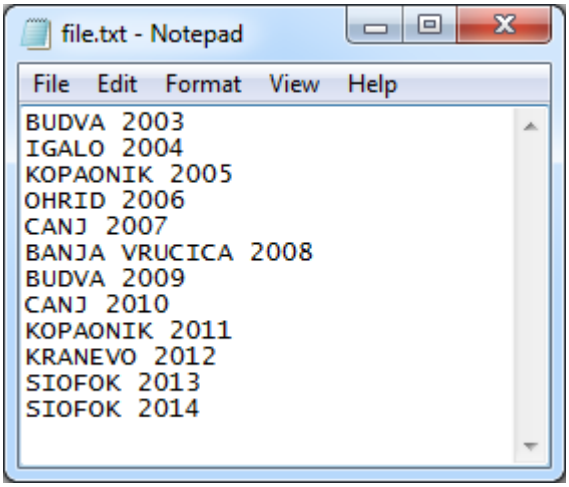
❼ **Find the output of the following program:**

```
#include <stdio.h>
#include <stdarg.h>
#define F(a,b) a##b

const char* f1(int, ...);
const char* f2(int, ...);

typedef const char*(*f_t)(int x, ...);
f_t f[3] = {F(f,1),F(f,2)};

const char* f1(int x, ...)
  { return __func__; }
int do_work()
  { return printf("%s", __func__);}

const char* f2(int x, ...)
{
    va_list va; va_start(va, x);
    return va_arg(va, f_t)(x);
}
const char* func(int w, ...)
{
  if(w)
  {
    va_list va; va_start(va, w);
    return va_arg(va,f_t)(1-w,f[1-w]);
  }
  else { int do_work(); return NULL; }
}

int main()
{
  for(int i=1; i>-1; i--)
     if (func(i,f[i]))
        printf("%s\n", func(i,f[i]));
  return 0;
}
```

❽ Find the output of the following program:

```
#include <stdio.h>
#include <stdlib.h>
#define GT(a,b) strcmp(a,b)
typedef struct node {char s[2][20];} NODE;

int main()
{
  NODE* f=NULL, x;
  char w[100];
  int i,j,k,n=-1;
  FILE *fp=fopen("file.txt", "r");
  while (fgets(w,100,fp))
  {
     if (f)
        realloc(f,(n+2)*sizeof(NODE));
     else
        f=(NODE*)malloc(sizeof(NODE));
     for (i=j=k=0, n++; w[j]>=' '; j++, k++)
     {
        if (!i&&w[j]=='2')
           (n+f)->s[i][--k]=0, i=1, k=0;
        (n+f)->s[i][k]=w[j];
        (n+f)->s[i][k+1]=0;
     }
  }
  for (i=1; i<=n; i++)
  {
    x=f[i];
    for (j=i; j>0&&GT(x.s[0],f[j-1].s[0]); )
       f[j]=f[j-1], j--;
    f[j]=x;
  }
  printf("%s\n", f[0].s[1]);
  printf("%s", f[n].s[1]);
  free(f);
  return 0;
}
```

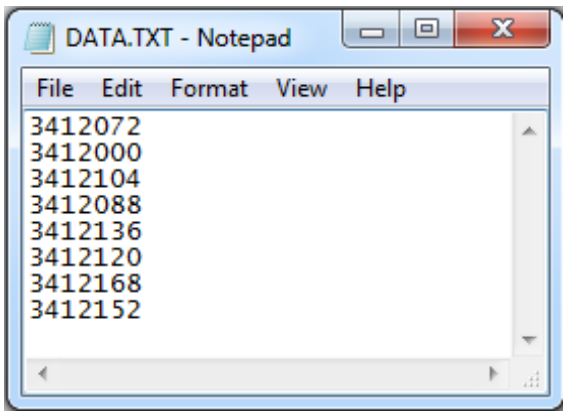
Assume the following input file:



❾ Find the output (printed on the stdout) of the following program:

```
#include <stdio.h>
#include <stdlib.h>
int comp(const void *x, const void *y)
  { return *(int**)x-*(int**)y; }
int **f(int n)
{
  int i;
  int **ppi=(int **)calloc(n,sizeof(int*));
  for (i=0; i<n; i++)
  {
    int *pc=(int*)malloc(sizeof(int));
    int *pi=(int*)malloc(sizeof(int));
    free(pc);
    if (i%2)
      while (pi>=*(ppi+i-1)) free(pi),
        pi=(int*)malloc(sizeof(int));
    else
      while (pi<=*(ppi+i+1)) free(pi),
        pi=(int*)malloc(sizeof(int));
    *pi=i; *(ppi+i)=pi;
  }
  return ppi;
}

int main()
{
  int i, **ppi = f(8);
  FILE *fp=fopen("DATA.TXT","w");
  for (i=0;i<8;i++)
     fprintf(fp,"%d\n",*(ppi+i));
  fclose(fp);
  qsort(ppi,8,sizeof(int*),comp);
  for (i=1; i<8; i++)
   if (**(ppi+i)>**(ppi+i-1))
     printf("%d-%d\n",**(ppi+i-1),**(ppi+i));
  return 0;
}
```

Assume the following output file DATA.TXT generated during the program execution:



⑩ Suppose that some project contains the following three source files, and find the output of the corresponding executable:

```
/* file: main.c */

#include <stdio.h>
extern int receiver(int);
int main()
{
   printf("%d", receiver(100));
   return 0;
}
```

```
/* file: sender.c */

#include <stdlib.h>
#define EXP(n) rand()%(n-1)
static int data[100], res;
int sender(int n)
{
   do res=EXP(n)+1; while (data[res]);
   return data[res]=res;
}
```

```
/* file: receiver.c */

extern int sender(int);
static int data[100], i, s;
int receiver(int n)
{
   for (i=1; i<n; i++)
      data[i]=sender(n);
   for (s=0, i=1; i<n; s+=data[i++]);
   return n*(n+1)/2-s;
}
```

Appendix A: ASCII table

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

Appendix B: Standard qsort() function

```
void qsort(void* base, size_t num, size_t size,
        int (*comp)(const void*,const void*));
```

The standard **qsort**() function sorts the **num** elements of the array pointed to by **base**, each element **size** bytes long, using the **comp()** function to determine the order. The **comp()** function follows the same comparison template as the standard **strcmp()** function.

Appendix C: Standard rand() function

```
int rand(void);
```

The standard **rand()** function returns a pseudo-random number in the range of 0 to RAND_MAX. RAND_MAX is a constant whose default value may vary between implementations but it is granted to be at least 32767.