编译原理第三次实验测试用例：目录

1　A 组测试用例

本组测试用例共 5 个，均为比较简单的程序，简单检查针对赋值/算术语句、分支语句、循环语句、数组表达式和函数调用的翻译。

1.1　A-1

1.1.1　输入

```c
int main() {
  int a = 10, b = 20, c, d, e, f, g, h, i, j;

  c = a + b;
  d = b - a;
  e = a * b;
  f = b / a;

  g = (b + a) * (b - a);
  h = (a + b) * (c - d);
  i = e / (f + g);
  j = (a * b) + (c / d) - (e + f);

  write(c);
  write(d);
  write(e);
  write(f);
  write(g);

  write(h);
  write(i);
  write(j);

  return 0;
}
```

1.1.2　输出

```
>>> Empty
[30, 10, 200, 2, 300, 600, 0, 1]
```

1.1.3 说明

这个测试用例针对赋值与算术语句进行测试。

输出中以>>>起始的行表示程序输入，其后的一行表示程序输出。预期输入、输出中每个数字会占一行，这里为了节省空间写在同一行（下同）。

1.2 A-2

1.2.1 输入

```
int main() {
  int a, b, c, d, e;
  int f = 60, g = 70;
  a = read();
  b = read();
  c = read();
  d = read();
  e = read();

  if (a < b) {
    write(a);
  }

  if (c > b) {
    write(b);
  } else {
    write(c);
  }

  if (d > a) {
    if (d > b) {
      write(d);
    } else {
      write(b);
    }
  } else {
    write(a);
  }

  if (e < a)
    write(1);
  else if (e < b)
    write(3);
  else if (e < c)
    write(5);
  else if (e < d)
    write(7);
  else
    write(9);

  if ((f > a) && (g < b))
    write(2);
  else if ((f > a) || (g < b))
    write(4);
  else
    write(6);

  return 0;
}
```

1.2.2　输出

```
>>> -3, 46, 1, 75, 111
[-3, 1, 75, 9, 4]

>>> 35, 21, 2, -8, -31
[2, 35, 1, 4]

>>> 1, -7, 12, 85, 101
[-7, 85, 9, 4]
```

1.2.3　说明

主要针对分支语句进行测试。

1.3 A-3

1.3.1 输入

```
int main() {
  int arr[10];
  int start, end, result = 0;
  int i;

  i = 0;
  while (i < 10) {
    arr[i] = i * i;
    i = i + 1;
  }

  start = read();
  end = read();

  i = start;
  while (i <= end) {
    result = result + arr[i];
    i = i + 1;
  }
  write(result);

  return 0;
}
```

1.3.2 输出

```
>>> 0, 1
[1]

>>> 5, 9
[255]
```

1.3.3 说明

主要测试一维数组。

1.4 A-4

1.4.1 输入

```
int calculateSum(int start, int end) {
  int sum = 0;
  int i = start;

  while (i <= end) {
    sum = sum + i;
    i = i + 1;
  }

  return sum;
}

int main() {
  int a, b, result;
  a = read();
  b = read();

  result = calculateSum(a, b);
  write(result);

  return 0;
}
```

1.4.2 输出

```
>>> 1, -1
[0]

>>> 0, 0
[0]

>>> 1, 10
[55]
```

1.4.3　说明

主要测试循环语句。

1.5　A-5

1.5.1　输入

```
int multiplyNumbers(int n) {
  if (n >= 1)
    return n * multiplyNumbers(n - 1);
  else
    return 1;
}

int main() {
  int x;
  x = read();
  write(multiplyNumbers(x));
  return 0;
}
```

1.5.2　输出

```
>>> 0
[1]

>>> 6
[720]
```

1.5.3　说明

主要测试函数调用。

2　B 组测试用例

本组测试用例共 3 个，较 A 组测试用例复杂，这里不专门针对赋值和算术语句设计测试用例。

2.1 B-1

2.1.1 输入

```
int binaryToDecimal(int binary) {
  int decimal = 0;
  int multiplier = 1;
  while (binary != 0) {
    decimal = decimal + (binary - (binary / 10) * 10) * multiplier;
    binary = binary / 10;
    multiplier = multiplier * 2;
  }
  return decimal;
}

int calculate_power(int base, int power) {
  int i = 0, result = 1;
  while (i < power) {
    result = result * base;
    i = i + 1;
  }
  return result;
}

int decimal_to_binary(int d) {
  int digitCount = 1;
  int j = digitCount - 1;
  int temp = d;
  while (temp >= 2) {
    temp = temp / 2;
    digitCount = digitCount + 1;
  }
  while (j >= 0) {
    if (d >= calculate_power(2, j)) {
      write(1);
      d = d - calculate_power(2, j);
    } else
      write(0);
    j = j - 1;
  }
  return 0;
}

int main() {
  int bin;
  int dec;

  bin = read();
  dec = binaryToDecimal(bin);
  write(dec);

  dec = read();
  decimal_to_binary(dec);

  return 0;
}
```

2.1.2 输出

```
>>> 1101, 16
[13, 1]
```

2.2 B-2

2.2.1 输入

```
int sumOfDigits(int num) {
  int digit;
  int sum = 0;
  while (num > 0) {
    digit = num - (num / 10) * 10;
    sum = sum + digit;
    num = num / 10;
  }
  return sum;
}
```

```

int sumOfSumsOfDigits(int s, int e) {
  int totalSum = 0, i = s;
  while (i <= e) {
    totalSum = totalSum + sumOfDigits(i);
    i = i + 1;
  }
  return totalSum;
}

int main() {
  int start, end, result;
  start = read();
  end = read();
  result = sumOfSumsOfDigits(start, end);
  write(result);
  return 0;
}
```

2.2.2 输出

```
>>> 10, 20
[57]
```

2.3 B-3

2.3.1 输入

```

int main() {
  int a[10], b[10], c[10], n;
  int i;
  n = read();

  i = 0;
  while (i < n) {
    a[i] = read();
    i = i + 1;
  }
  i = 0;
  while (i < n) {
    b[i] = read();
    i = i + 1;
  }

  i = 0;
  while (i < n) {
    c[i] = a[i] * b[i];
    i = i + 1;
  }

  i = 0;
  while (i < n) {
    write(c[i]);
    i = i + 1;
  }

  return 0;
}
```

2.3.2 输出

```
>>> 2, 1, 1, 1, 1
[1, 1]

>>> 4, 3, 2, 1, 0, 2, 4, 5, 7
[6, 8, 5, 0]
```

3 C 组测试用例

本组测试用例共 2 个，是较经典的问题。

3.1 C-1

3.1.1 输入

```
int min(int a, int b) {
  if (a <= b)
    return a;
  else
    return b;
}

int jumpSearch() {
  int arr[16];
  int x;
  int n = 16;
  int step = 4;
  int i = 0;
  int prev = 0;

  while (i < 16) {
    arr[i] = read();
    i = i + 1;
  }
  x = read();

  while (arr[min(step, n) - 1] < x) {
    prev = step;
    step = step + 4;
    if (prev >= n)
      return -1;
  }

  while (arr[prev] < x) {
    prev = prev + 1;

    if (prev == min(step, n))
      return -1;
  }
  if (arr[prev] == x)
    return prev;

  return -1;
}

int main() {
  int index = jumpSearch();
  write(index);
  return 0;
}
```

3.1.2 输出

```
>>> 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 55
[10]

>>> 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 610
[15]
```

3.2 C-2

3.2.1 输入

```
int min(int a, int b) {
  if (a <= b)
    return a;
  else
    return b;
}

int fibMonaccianSearch() {
  int arr[16];
  int x;
  int i = 0;
  int n = 16;
  int offset = -1;

  int fibMMm2 = 0;
  int fibMMm1 = 1;
  int fibM = fibMMm2 + fibMMm1;
```

```

while (i < 16) {
  arr[i] = read();
  i = i + 1;
}
x = read();

while (fibM < n) {
  fibMMm2 = fibMMm1;
  fibMMm1 = fibM;
  fibM = fibMMm2 + fibMMm1;
}

while (fibM > 1) {
  i = min(offset + fibMMm2, n - 1);

  if (arr[i] < x) {
    fibM = fibMMm1;
    fibMMm1 = fibMMm2;
    fibMMm2 = fibM - fibMMm1;
    offset = i;
  }

  else if (arr[i] > x) {
    fibM = fibMMm2;
    fibMMm1 = fibMMm1 - fibMMm2;
    fibMMm2 = fibM - fibMMm1;
  }

  else
    return i;
}

if (fibMMm1 && arr[offset + 1] == x)
  return offset + 1;

return -1;
```

```
}

int main() {
  write(fibMonaccianSearch());
  return 0;
}
```

3.2.2　输出

```
>>> 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 55
[10]

>>> 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 610
[15]
```

4　E 组测试用例

本组测试用例共 6 个，针对不同分组进行测试。

E1 组针对 3.1 分组测试结构体的翻译，E2 组针对 3.2 分组测试一维数组作为参数和高维数组的翻译。每组 3 个测试用例。

4.1　E1-1

4.1.1　输入

```

struct Activity {
  int start;
  int finish;
};

int main() {
  struct Activity a1, a2;
  a1.start = read();
  a1.finish = read();
  a2.finish = a1.start;
  a2.start = a1.finish;
  write(a1.start);
  write(a1.finish);
  write(a2.start);
  write(a2.finish);
  return 0;
}
```

4.1.2 输出

```
>>> 1, 2
[1, 2, 2, 1]
```

4.1.3 说明

测试对于简单结构体的翻译。针对选做**要求 4.1** 的同学，其他同学需要提示无法翻译且不输出中间代码。

4.2 E1-2

4.2.1 输入

```
struct Rectangle {
  int length;
  int width;
};

int calculateArea(struct Rectangle rect) { return rect.length * rect.
   width; }

int main() {
  int numRectangles;
  int totalArea = 0;
  struct Rectangle r;

  numRectangles = read();
  while (numRectangles > 0) {
    r.length = read();
    r.width = read();
    totalArea = totalArea + calculateArea(r);
    numRectangles = numRectangles - 1;
  }

  write(totalArea);
  return 0;
}
```

4.2.2 输出

```
>>> 2, 3, 4, 7, 8
[68]
```

4.2.3 说明

测试将结构体作为函数参数。针对选做**要求 4.1** 的同学，其他同学需要提示无法翻译且不输出中间代码。

4.3 E1-3

4.3.1 输入

```

struct Point {
  int x;
  int y;
};

struct Circle {
  struct Point center;
  int radius;
};

int calculateCircleArea(struct Circle circle) {
  return 3 * circle.radius * circle.radius;
}

int sumOfAreasPlusDistanceSquared(struct Circle circle1,
                                  struct Circle circle2) {
```

```c
  int area1 = calculateCircleArea(circle1);
  int area2 = calculateCircleArea(circle2);
  int sumOfAreas = area1 + area2;

  int dx = circle1.center.x - circle2.center.x;
  int dy = circle1.center.y - circle2.center.y;
  int distance = dx * dx + dy * dy;

  return sumOfAreas + distance;
}

int main() {
  struct Circle c1, c2;

  c1.center.x = read();
  c1.center.y = read();
  c1.radius = read();

  c2.center.x = read();
  c2.center.y = read();
  c2.radius = read();

  write(sumOfAreasPlusDistanceSquared(c1, c2));

  return 0;
}
```

4.3.2 输出

```
>>> 1, 3, 5, 2, 4, 6
[185]
```

4.3.3 说明

测试对于较复杂的结构体及其作为函数参数进行函数的调用。针对选做**要求 4.1** 的同学，其他同学需要提示无法翻译且不输出中间代码。

4.4 E2-1

4.4.1 输入

```
int main() {
  int n = 3;
  int magicSquare[3][3];
  int i = n / 2;
  int j = n - 1;
  int num = 1;

  while (num <= n * n) {
    if (i == -1 && j == n) {
      j = n - 2;
      i = 0;
    } else {
      if (j == n)
        j = 0;
      if (i < 0)
        i = n - 1;
    }
    if (magicSquare[i][j]) {
      j = j - 2;
      i = i + 1;
    } else {
      magicSquare[i][j] = num;
      num = num + 1;
      j = j + 1;
      i = i - 1;
    }
  }
  write(n);
  write(n * (n * n + 1) / 2);
  i = 0;
  j = 0;
  while (i < n) {
    while (j < n) {
      write(magicSquare[i][j]);
      j = j + 1;
    }
    i = i + 1;
  }
  return 0;
}
```

4.4.2 输出

```
>>> Empty
[3, 15, 2, 7, 6]
```

4.4.3 说明

测试对于简单高维数组的翻译，不涉及数组作为函数参数。针对选做**要求 4.2** 的同学，其他同学需要提示无法翻译且不输出中间代码。

4.5 E2-2

4.5.1 输入

```
int maxSumarray(int a[10], int s) {
  int j = 0;
  int max_sum_so_far = 0;
  int max_ending_here = 0;

  while (j < s) {
    max_ending_here = max_ending_here + a[j];

    if (max_ending_here < 0) {
      max_ending_here = 0;
    }
    if (max_sum_so_far < max_ending_here) {
      max_sum_so_far = max_ending_here;
    }
    j = j + 1;
  }
  return max_sum_so_far;
}

int main() {

  int i = 0, size = 10;
  int arr[10];
  while (i < size) {
    arr[i] = read();
    i = i + 1;
  }
  write(maxSumarray(arr, size));
  return 0;
}
```

4.5.2 输出

```
>>> 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
[143]
```

4.5.3 说明

测试对于数组作为函数参数的翻译。针对选做**要求 4.2** 的同学，其他同学需要提示无法翻译且不输出中间代码。

4.6 E2-3

4.6.1 输入

```
int minDistance(int n, int visited[10], int d[10]) {
  int min = 2147483647, min_vertex;
  int v = 0;
  while (v < n) {
    if (!visited[v] && d[v] <= min) {
      min = d[v];
      min_vertex = v;
    }
    v = v + 1;
  }
  return min_vertex;
}

int main() {
  int i, j, k;
  int graph[10][10], isvisited[10], dist[10];
  int sizeOfGraph, Edges, minV;
  sizeOfGraph = read();
  i = 0;
  j = 0;
  while (i < sizeOfGraph) {
    while (j < sizeOfGraph) {
      graph[i][j] = 0;
      j = j + 1;
    }
    i = i + 1;
  }
  Edges = read();
  i = 0;
  while (i < Edges) {
    j = read();
    k = read();
    graph[j][k] = read();
    graph[k][j] = graph[j][k];
    i = i + 1;
  }

  i = 0;
  while (i < sizeOfGraph) {
    dist[i] = 2147483647;
    isvisited[i] = 0;
    i = i + 1;
  }

  dist[0] = 0;

  i = 0;
  while (i < sizeOfGraph - 1) {
    minV = minDistance(sizeOfGraph, isvisited, dist);
    isvisited[minV] = 1;

    j = 0;
    while (j < sizeOfGraph) {
      if (isvisited[j] == 0 && graph[minV][j] &&
          dist[minV] + graph[minV][j] < dist[j])
        dist[j] = dist[minV] + graph[minV][j];
      j = j + 1;
    }
    i = i + 1;
  }

  i = 0;
  while (i < sizeOfGraph) {
    write(dist[i]);
    i = i + 1;
  }
  return 0;
}
```

4.6.2 输出

```
>>> 4, 5, 0, 1, 2, 0, 2, 1, 1, 2, 6, 1, 3, 1, 2, 3, 3
[0, 2, 1, 3]
```

4.6.3 说明

测试对于较复杂的数组及其作为函数参数进行函数的调用。针对选做**要求 4.2** 的同学，其他同学需要提示无法翻译且不输出中间代码。

5 结束语

若对本文档有任何疑议，可写邮件与周意可助教联系，注意同时抄送给许畅老师。