COMPUTERIZED LAYOUT PLANNING (COMPLAN)

by

Mr. Wit Watcharodomprasert

A Thesis of the Twelve-Credit Course
CE 7000 Master Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer and Engineering Management
Assumption University

April 2001
COMPUTERIZED LAYOUT PLANNING (COMPLAN)

by

Mr. Wit Watcharodomprasert

A Thesis of the Twelve-Credit Course
CE 6998 Master Thesis

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Computer and Engineering Management
Assumption University

April 2001
The Graduate School of Assumption University has approved this final report of the twelve-credit course, CE 7000 MASTER THESIS, submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer and Engineering Management.

Approval Committee:

(Dr. Chamnong Jungthirapanich)  
Dean and Advisor

(Prof. Dr. Srisakdi Charmonman)  
Chairman

(Asst. Prof. Dr. Boonmark Sirinavakul)  
Member

(Assoc. Prof. Somchai Thayaranyak)  
MUA Representative

April 2001
ABSTRACT

This thesis attempts to develop a prototype of a computerized layout planning software, named COMPLAN (Computerized Layout Planning). COMPLAN assists layout planners in making effective and timely decisions on facilities layout planning.

Various layout planning theories and techniques were investigated. The knowledge and understanding obtained through the investigation helps identify the most appropriate layout planning methodology for use in the development of COMPLAN. Since the introduction of the first computerized layout planning software in 1964, several computer-based layout planning tools have been proposed to further enhance the capabilities of their precursors. However, these packages still have a number of shortcomings, resulting in limited applicability, and questionable fidelity of the layouts generated.

COMPLAN was developed, considering capabilities and limitations of the previous systems. The prototype employs a graphical user interface (GUI) to enhance the system's ease of use. Delphi was selected as the system's programming language, since it can be operated on Microsoft Windows, a famous and widely used operating system. The program analyzes, designs, and evaluates facilities layouts via the closeness relationships. Practical layout plans are generated and displayed in graphics. Evaluation of alternatives and selection of the most appropriate layouts can be accomplished through weighted factor scoring.
ACKNOWLEDGEMENTS

I had taken a great effort on this thesis paper for a long period of time and now it has finished with a lot of assists from these people.

First, I would like to thank Dr. Chamnong Jungthirapanich, the advisor of this thesis for his deeply continuous advice through a long period and Mr. Satit Thongjeen for help with a lot of programming techniques.

Next, many others have contributed directly and indirectly to me like giving me some interesting guide books and relieving some parts from my job. I call them “colleagues and friends”. They have extended encouragement and support. Last but not least, I also would like to offer a special thanks to my beloved parents, my brothers, and Ms. Narawadee Siengsung, who are always waiting for my success.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>i</td>
</tr>
<tr>
<td>ACTKNOWLEDGEMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>viii</td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. LITERATURE REVIEW</td>
<td>4</td>
</tr>
<tr>
<td>2.1 Manufacturing Facility Layout</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Layout Planning and Methodology</td>
<td>7</td>
</tr>
<tr>
<td>2.3 Computerized Layout Planning Technique</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Computerized Layout Planning Programs</td>
<td>15</td>
</tr>
<tr>
<td>2.5 Software Development Process</td>
<td>27</td>
</tr>
<tr>
<td>III. SYSTEM DEVELOPMENT</td>
<td>33</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>33</td>
</tr>
<tr>
<td>3.2 COMPLAN Flowchart</td>
<td>34</td>
</tr>
<tr>
<td>3.3 Layouts Development Algorithm</td>
<td>48</td>
</tr>
<tr>
<td>3.4 Layouts Improvement Algorithm</td>
<td>52</td>
</tr>
<tr>
<td>3.5 Layouts Evaluation</td>
<td>53</td>
</tr>
<tr>
<td>3.6 Data Requirements</td>
<td>54</td>
</tr>
<tr>
<td>3.7 Peripheral Requirements</td>
<td>54</td>
</tr>
<tr>
<td>3.8 Graphical User Interface</td>
<td>54</td>
</tr>
<tr>
<td>IV. SYSTEM EVALUATION</td>
<td>56</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>56</td>
</tr>
<tr>
<td>4.2 Verification</td>
<td>57</td>
</tr>
<tr>
<td>Chapter</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------</td>
</tr>
<tr>
<td>4.3 Validation</td>
<td>60</td>
</tr>
<tr>
<td>V. CONCLUSIONS &amp; RECOMMENDATIONS</td>
<td>68</td>
</tr>
<tr>
<td>5.1 Conclusions</td>
<td>68</td>
</tr>
<tr>
<td>5.2 Recommendations</td>
<td>71</td>
</tr>
<tr>
<td>5.3 Future Research</td>
<td>71</td>
</tr>
<tr>
<td>APPENDIX A COMPLAN'S MANUAL</td>
<td>73</td>
</tr>
<tr>
<td>APPENDIX B COMPLAN'S SOURCE CODES</td>
<td>87</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>139</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 A Process Layout</td>
<td>4</td>
</tr>
<tr>
<td>2.2 A Product Layout</td>
<td>5</td>
</tr>
<tr>
<td>2.3 A Fixed-position Layout</td>
<td>5</td>
</tr>
<tr>
<td>2.4 Ideal Systems Hierarchy</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Systematic Layout Planning Procedure</td>
<td>12</td>
</tr>
<tr>
<td>2.6 An Example of a Layout Generated via CRAFT</td>
<td>17</td>
</tr>
<tr>
<td>2.7 An Example of a Layout Generated via COFAD</td>
<td>20</td>
</tr>
<tr>
<td>2.8 An Example of a Layout Generated via PLANET</td>
<td>23</td>
</tr>
<tr>
<td>2.9 An Example of a Layout Generated via CORELAP</td>
<td>24</td>
</tr>
<tr>
<td>2.10 Overall Sweep Pattern Followed by ALDEP</td>
<td>26</td>
</tr>
<tr>
<td>2.11 An Example of a Layout Generated via ALDEP</td>
<td>27</td>
</tr>
<tr>
<td>2.12 Software Testing Strategy</td>
<td>28</td>
</tr>
<tr>
<td>2.13 Software Testing Steps</td>
<td>28</td>
</tr>
<tr>
<td>2.14 White Box Testing</td>
<td>32</td>
</tr>
<tr>
<td>2.15 Black Box Testing</td>
<td>32</td>
</tr>
<tr>
<td>3.1 Overall COMPLAN Flowchart</td>
<td>35</td>
</tr>
<tr>
<td>3.2 An Example of Inputting Step of COMPLAN</td>
<td>37</td>
</tr>
<tr>
<td>3.3 The Relationship Diagram of COMPLAN</td>
<td>37</td>
</tr>
<tr>
<td>3.4 Calculate TCR Procedure</td>
<td>39</td>
</tr>
<tr>
<td>3.5 An Example of Calculating TCR in COMPLAN</td>
<td>40</td>
</tr>
<tr>
<td>3.6 Queue Activity Area Procedure</td>
<td>41</td>
</tr>
<tr>
<td>3.7 Locate Activity Area Procedure</td>
<td>42</td>
</tr>
<tr>
<td>3.8 An One Unit Block Layout in COMPLAN</td>
<td>43</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.9</td>
<td>Improve Layout Diagram Procedure</td>
</tr>
<tr>
<td>3.10</td>
<td>Draw Layout Procedure</td>
</tr>
<tr>
<td>3.11</td>
<td>A Layout Generated in COMPLAN</td>
</tr>
<tr>
<td>3.12</td>
<td>An Evaluation Score in COMPLAN</td>
</tr>
<tr>
<td>3.13</td>
<td>Calculate Evaluated Score Procedure</td>
</tr>
<tr>
<td>3.14</td>
<td>A Locating Method by COMPLAN</td>
</tr>
<tr>
<td>3.15</td>
<td>Illustration of Delphi 5.0</td>
</tr>
<tr>
<td>4.1</td>
<td>Test Information Flow</td>
</tr>
<tr>
<td>4.2</td>
<td>Error Message</td>
</tr>
<tr>
<td>4.3</td>
<td>The Recommend Result via Expert with 29 Evaluated Score</td>
</tr>
<tr>
<td>4.4</td>
<td>The Result via BLOCPLAN-90 with 33 Evaluated Score</td>
</tr>
<tr>
<td>4.5</td>
<td>The Result via COMPLAN with 34 Evaluated Score</td>
</tr>
<tr>
<td>4.6</td>
<td>The Recommend Result via Expert with 18 Evaluated Score</td>
</tr>
<tr>
<td>4.7</td>
<td>The Result via BLOCPLAN-90 with 21 Evaluated Score</td>
</tr>
<tr>
<td>4.8</td>
<td>The Result via COMPLAN with 21 Evaluated Score</td>
</tr>
<tr>
<td>A.1</td>
<td>Illustration of COMPLAN</td>
</tr>
<tr>
<td>A.2</td>
<td>Illustration of File Menu</td>
</tr>
<tr>
<td>A.3</td>
<td>Illustration of Print Menu</td>
</tr>
<tr>
<td>A.4</td>
<td>Illustration of View Menu</td>
</tr>
<tr>
<td>A.5</td>
<td>Illustration of TCR Table</td>
</tr>
<tr>
<td>A.6</td>
<td>Illustration of Config Menu</td>
</tr>
<tr>
<td>A.7</td>
<td>Illustration of Step 1 Establish Space Requirements</td>
</tr>
<tr>
<td>A.8</td>
<td>Illustration of Step 2 Chart the Relations</td>
</tr>
<tr>
<td>A.9</td>
<td>Illustration of Step 3 Diagram Activity Relationships</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>A.10 Illustration of Step 4 Draw Space Relationship Layout</td>
<td>84</td>
</tr>
<tr>
<td>A.11 Two Additional Menu on Right Mouse</td>
<td>84</td>
</tr>
<tr>
<td>A.12 Illustration of Zoom Function</td>
<td>85</td>
</tr>
<tr>
<td>A.13 Illustration of Step 5 Evaluate Alternative Arrangements</td>
<td>86</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>50</td>
</tr>
<tr>
<td>4.1</td>
<td>62</td>
</tr>
<tr>
<td>4.2</td>
<td>63</td>
</tr>
<tr>
<td>4.3</td>
<td>65</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

Facilities layout and location are the problems that always occur in industrial development. Whenever a new plant is built, the planner must do the layout planning. These problems can be separated into 2 categories, location problem and facility layout planning problem. Location problem includes outside plant problems such as finding new plant location, and inside plant problems such as finding the location of machine or equipment. There are a lot of concerned criteria to be considered in finding the new location such as resources, market, transportation and etc. Another problem, facility layout planning is concerned the management of space area, machines and other equipment to be appropriate with process flow and other conditions inside the factory.

The common criteria used to be considered in layout planning are utilizing existing space most effectively, maintain flexibility of arrangement and operation, facilitate the manufacturing process, provide for employee convenience, safety, and comfort, and minimize material handling cost.

Layout planning is a major problem in designing industrial plant due to it being impossible to satisfy all criteria. Material handling cost is directly concerned with the whole plant cost then to minimize material handling cost always be considered as majority and other criteria is to be considered as minority.

In industrial development, most producers pay attention only to expanding line production and they are not interested in improving efficiency. This usually leads to higher capital investment. In the competitive world, every possible avenue to cost reduction must be sought, quality, cost, and efficiency are of utmost importance.

With good layout planning, all resources such as workers, machines, materials, space, and time are used at the highest efficiency. Materials are moved quickly and effectively. It helps reduce such costs as transportation costs, materials handling costs.
Designing appropriate layout plans is a way to improve productivity and increase profits, since poor layout planning is a drain of numerous operations losses. Thus, layout planning is the method that can be used to both decreasing capital requirements and increasing productivity.

There are several methods and techniques that can solve this kind of problem. But most of the problems are large, a lot of huge data, more complicated, more considered factors and have more possible results. Therefore, computer is considered as an assistant equipment to solve these kinds of problems. Using computer in solving the layout problem has several advantages over the traditional manual approach. First, the computer can perform the necessary calculations and generate several solutions much more quickly than manual procedures. Second, the computer can solve large-sized problems, which usually involve huge amounts of data. Third, since the computer can develop solutions quickly, it is more economical to use a computer as an aid in the design process than to depend on human planners and designers alone. And fourth, by using the computer, solutions will be developed on the basis of mathematical expressions and operations that can be evaluated objectively. These solutions are often better than those produced solely by subjective judgement. (Sule 1994)

With previous explanation, layout planning problem is a major topic in this proposal. This thesis proposal attempts to develop computer software, named COMPLAN, for solving facility layout planning problems. In designing a layout plan, analyses of a large number of departmental relationships to find the appropriate layout is often time-consuming and complicated. The program helps reduce time spent on analyzing and designing layouts. The program also displays the layout scores of all layouts that are generated. The layout scores facilitate layout planners in making decisions on the most appropriate layouts.
The thesis plan begins with studying theories and methodology of facilities layout planning, learning the Delphi programming language, designing software, writing, and testing the program, and creating a user's manual.
II. LITERATURE REVIEW

2.1 Manufacturing Facilities Layouts

There are three basic types of manufacturing facilities layouts including process layouts, product layouts, and fixed-position layouts. (Gaither 1996)

1) Process layouts are layouts planned by categorizing the same types of machines into groups or departments. They are suitable for job shops which produce many kinds of goods. Figure 2.1 illustrates the process layouts.

![Figure 2.1 - A Process Layout](image)

(2) Product layouts are layouts planned by following the process flows. This type of layout is suitable for mass production which has a fixed process flow. Figure 2.2 illustrates the product layouts.
(3) Fixed-position layouts are layouts planned for producing large, heavy or bulky products such as aircrafts, missiles, or ships. All tools and machines are transported to the product. Figure 2.3 illustrates the fixed-position layouts.

Process layouts, product layouts and fixed-position layouts are compared in Table 2.1. These following tables itemize advantages and limitations of different types of layouts. (Tompkins et al. 1996)
Table 2.1. Advantages and Limitations of Process Layouts.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Increased machine utilization.</td>
<td>1. Increased material handling requirements.</td>
</tr>
<tr>
<td>2. General-purpose equipment can be used.</td>
<td>2. More complicated production control required.</td>
</tr>
<tr>
<td>3. Highly flexible in allocating personnel and equipment.</td>
<td>3. Increased work-in-process.</td>
</tr>
<tr>
<td>4. Lower equipment investment.</td>
<td>4. Longer production times.</td>
</tr>
<tr>
<td>5. Diversity of tasks for personnel.</td>
<td>5. Higher skills required to accommodate diversity of tasks required.</td>
</tr>
<tr>
<td>6. Specialized supervision is possible.</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. Advantages and Limitations of Product Layouts.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Smooth, simple, logical, and direct flow line result.</td>
<td>1. Machine stoppage stops the line.</td>
</tr>
<tr>
<td>2. Small work-in-process inventories should result.</td>
<td>2. Product design changes cause the layout to become obsolete.</td>
</tr>
<tr>
<td>3. Total production time per unit is short.</td>
<td>3. Slowest station paces the line.</td>
</tr>
<tr>
<td>4. Material handling requirements are reduced</td>
<td>4. General supervision is required.</td>
</tr>
<tr>
<td>5. Less skill is required for personnel</td>
<td>5. Higher equipment investment usually results.</td>
</tr>
<tr>
<td>6. Simple production control is possible.</td>
<td></td>
</tr>
<tr>
<td>7. Special-purpose equipment can be used.</td>
<td></td>
</tr>
</tbody>
</table>
Table 2.3. Advantages and Limitations of Fixed-position Layouts.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Material movement is reduced</td>
<td>1. Personal and equipment movement is increased.</td>
</tr>
<tr>
<td>2. When a team approach is used, continuity of operations and responsibility results.</td>
<td>2. May result in duplicate equipment.</td>
</tr>
<tr>
<td>3. Provides job enrichment opportunities.</td>
<td>3. Requires greater skills for personnel.</td>
</tr>
<tr>
<td>4. Promotes pride and quality because and individual can complete the “whole job.”</td>
<td>4. Requires general supervision.</td>
</tr>
<tr>
<td>5. Highly flexible; can accommodate changes in product design, product mix, and production volume.</td>
<td>5. May result in increased space and greater work-in-process.</td>
</tr>
<tr>
<td>6. Requires close control and coordination in scheduling production.</td>
<td></td>
</tr>
</tbody>
</table>

2.2 Layout Planning and Methodology

A number of procedures have been developed to aid the facilities planners in designing layouts. Most of them followed the steps listed below (Tomkins and White 1984).

1. Define or redefine the objective of the facility.

2. Specify the primary and support activities to be performed in accomplishing the objective.

3. Determine the interrelationships among all activities.

4. Determine the space requirements for all activities.

5. Generate alternative layouts.

6. Evaluate alternative layouts.
(7) Select the best facility layout.

(8) Implement the selected layout.

(9) Maintain and adapt the installed layout, as appropriate.

However, there are several other procedures used in designing plant layouts (Tompkins and White 1984). Specifically contrast on the ideal systems approached by Immer, Nadler, Apple and Reed including systematic layout planning (SLP) and simplified systematic layout planning (Simplified SLP) of Muther.

(1) Immer's Basic Steps.

Three simple steps are:

(a) Put the problem on paper.

(b) Show lines of flow.

(c) Convert flow lines to machine lines.

(2) Nadler's Ideal Systems Approach.

Nadler identified his designing approach as the ideal systems approach. Figure 2.4 depicts the hierarchical approach.

(a) Aim for the "theoretical ideal system."

(b) Conceptualize the "ultimate ideal system."

(c) Design the "technologically workable ideal system."

(d) Install the "recommended system."
Apple’s Plant Layout Procedure.

Apple proposed the following detail sequence steps in designing layout.

(a) Procure the basic data.
(b) Analyze the basic data.
(c) Design the productive process.
(d) Plan the material flow pattern.
(e) Consider the general material-handling plan.
(f) Calculate equipment requirements.
(g) Plan individual workstations.
(h) Select specific material handling equipment.
(i) Coordinate groups of related operations.
(j) Design activity interrelationships.
(k) Determine storage requirements.

Figure 2.4. Ideal Systems Hierarchy (Francis, McGinnis, and White 1992).
(l) Plan service and auxiliary activities.

(m) Determine space requirements.

(n) Allocate activities to total space.

(o) Consider building types.

(p) Construct master layout.

(q) Evaluate, adjust, and check the layout with the appropriate persons.

(r) Obtain approvals.

(s) Install the layout.

(t) Follow up on implementation of the layout.

(4) Reed’s Plant Layout Procedure.

Reed recommended the required steps in planning layout.

(a) Analyze the product or products to be produced.

(b) Determine the process required manufacturing the product.

(c) Prepare layout-planning charts.

(d) Determine workstations.

(e) Analyze storage area requirements.

(f) Establish minimum aisle widths.

(g) Establish office requirements.

(h) Consider personnel facilities and services.

(i) Survey plant services.

(j) Provide for future expansion.

(5) Systematic Layout Planning.

The most popular approach used in designing plant layouts has been the systematic layout planning approach developed by Muther (Muther 1973). Referred to as SLP, the procedure has been applied to production,
transportation, storage, supporting services, and office activities, among others. The SLP procedure is depicted in Figure 2.5. Once the appropriate information is gathered, a flow analysis can be combined with an activity analysis to develop the relationship diagram. Space considerations, when combined with the relationship diagram, lead to the construction of the space relationship diagram. Based on the space relationship diagram, modifying considerations, and practical limitations, a number of alternative layouts are designed and evaluated. In comparison with the steps of the design process, SLP begins after the problem is formulated. The first five steps of SLP involve the analysis of the problem. Step 6 through 9, including the generation of alternative layouts, constitute the search phase of the design process. The selection phase of the design process coincides with step 10 of SLP.
Figure 2.5. Systematic Layout Planning Procedure (Muther 1973).
This Simplified SLP is a set of six procedures to follow when laying out an area. It is suited to smaller projects that do not require the full SLP treatment. Each of the six steps has its own symbol. The explanation for these symbols is as following:

(a) **Triangle** – A triangular-shaped relationship chart.
(b) **Square** - Square feet and physical features.
(c) **Star** – A diagram connecting activities at different points.
(d) **Circle** - Round and round to adjust the layout diagrammed.
(e) **Hexagon** - Examine from all sides; evaluate all factors.
(f) **Rectangle** – A layout plan on sheet of paper or building print.

The six steps are:

(a) Chart the relationships.

First step is to relate each activity, area, function, or major building feature involved in the contemplated layout to every other activity by a closeness-desired rating. This is a rating step to determine the relative closeness between each pair of activities or areas.

(b) Establish space requirements.

Requirements of each activity area are established, such as, the area, physical features and utilities, and other conditions.

(c) Diagram activity relationships.

Use the relationships of data to relate all various activities to each other visually and geographically to form the basic pattern for layout called relationship diagram.
(d) Draw space relationship layouts.

Visually and geographically arrange the space required for all activities. Any adjustments or rearrangements necessary to integrate all modifying considerations.

(e) Evaluate alternative arrangements.

Evaluate all layout arrangements according to the various factors to identify the best layout most suitable for company.

(f) Detail the selected layout plan.

Final step, draw all the detailed equipment or features on the selected layout. The completed plan can now be used to guide the installation.

2.3 Computerized Layout Planning Techniques

Apart from these manual procedures, layout-planning problems can also be solved by computerized approaches.

Following exemplify computerized layout planning techniques (Sule 1994).

(1) Mathematical procedures

The plant layout problem is formulated generally as a quadratic assignment problem (QAP), where “m” locations are available to which “n” departments are to be assigned. Most of these approaches are feasible only for small sized problem because the larger number of departments drastically increases possible combinations of solutions.

(2) Heuristics

This approach is capable of producing acceptable suboptimal solutions and has the capacity to handle large problems with a reasonable amount of computational effort.
(3) Probabilistic approaches

Probabilistic approaches, like exact mathematical procedures and heuristics, use known data. The probabilistic aspects stem from the manner in which departments are selected to enter the layout in construction procedures or interchange locations in improvement algorithms.

Construction procedures begin with the basic problem data and build up a solution, or permutation, in an iterative model. Initially, none of the activity area will have been assigned to a location. At any intermediate point in the construction process, the activity area has been assigned to a location.

Improvement algorithms modify a given solution to optimize the departmental relationship. When the solution represents a permutation, the simplest possible modification is to select two elements of the permutation and interchange their values.

(4) The graph theory

There are also a number of algorithms based on the graph theory in which the reach department is represented initially as a node. After planar graph is developed to identify adjacent departments, a heuristic procedure is applied to construct a block layout.

2.4 Computerized Layout Planning Programs

Computerized layout planning program can be categorized as construction programs, improvement programs or both. The following are examples of computerized layout programs developed (Sule 1994).
2.4.1 CRAFT

In 1963, the first computer-aided layout routine appeared. The technique, CRAFT, Computerized Relative Allocation of Facilities Technique, attempts to develop a layout that minimizes transportation cost, where transportation cost is defined as the product of from-to chart, move cost chart, and distance chart. CRAFT’s inputs include a from-to chart, a move cost chart and an initial layout. CRAFT requires move costs to be input in the form of cost per unit moved per unit distance. To input move costs in this manner requires the following assumptions:

1. Move costs are independent of the utilization of the equipment.
2. Move costs are linearly related to the length of the move.

The CRAFT procedure begins by determining the centroids of the departments in the initial layout. It then calculates the rectilinear distance between department centroids and stores the distance in a distance chart. The transportation cost for the initial layout is determined by calculating the product of the initially input from-to chart, move cost chart, and distance chart. CRAFT next considers departmental interchanges for departments that are of equal area or have a common border in an effort to reduce the transportation cost. The following types of interchanges can be considered.

1. Pair-wise interchanges.
2. Three-way interchanges.
3. Pair-wise followed by three-way interchanges.
4. Three-way followed by pair-wise interchanges.
5. The best of pair-wise or three-way interchanges.

The transportation cost is approximated for each departmental interchange. The departmental interchange offering the greatest reduction in transportation cost is made
to the layout. CRAFT continues by considering departmental interchanges to the new layout, approximating the transportation costs for these interchanges, and once again making the interchange that offers the greatest estimated reduction in transportation cost. The procedure continues until no interchanges in the layout can be found that reduces the transportation cost; the search is then terminated.

Figure 2.6. An Example of a Layout Generated via CRAFT (Tompkins and Moore 1980).

2.4.2 COFAD

COFAD, COnputerized FACilities Design, is basically a modification of CRAFT to allow the use of move costs for a variety of material handling equipment alternatives. To include such move costs, the material handling methods and costs must be determined for each alternative layout. COFAD is a model that jointly considers both the layout and the material handling system. The inputs required by COFAD are alternative materials handling equipment for performing specific moves, the costs of these alternatives, from-to charts for each equipment alternative, and an initial layout. The COFAD objective to jointly select a layout and the material handling system that
approaches the minimal-cost handling system. The iterative functioning of COFAD may be described by the following steps:

(1) Determine a layout.

(2) Select a material handling system.

(3) Apportion the costs of the handling system to the individual moves.

(4) Return to Step 1.

These four steps are repeated until a steady-state solution is reached at which point either the model is terminated or sensitivity analysis may be performed on the form-to charts.

The first function of COFAD is to improve the initial layout by a procedure that is almost identical to CRAFT. COFAD determines the costs of performing each move with the feasible material handling equipment alternatives. The method of determining the costs to perform each move is dependent on the type of material handling equipment used. For fixed-path equipment (conveyors, cranes, and hoists), the annual cost of each move is determined by performing the following calculation:

\[
\text{Move cost} = \text{Variable cost per meter} \times \text{Total annual distance of the move} + \text{Annual nonvariable cost.}
\]

Variable and nonvariable costs are initially input and the total annual distance of the move is calculated by COFAD. For mobile equipment (all equipments except for conveyors, cranes, and hoists), the move costs are determined by the following calculation:
Move cost = Variable cost per hour x Move time in hour per year
+ Annual nonvariable cost x Percent equipment utilization for this move

Variable and nonvariable costs are initially input and move time and equipment utilization are calculated by COFAD.

COFAD's next function is to utilize the move costs in an effort to determine a minimal-cost material handling system. The initially selected material handling system is determined by selecting the material handling equipment alternative for each move that has the lowest move cost for that move.

The first improvement is performed in an effort to increase the utilization of all equipment types. The utilizations for each assigned equipment type are summed and rounded up to determine the required quantity of each equipment type. These rounded numbers are referred to as the design equipment requirements for each equipment type. The differences between design equipment requirements and the sums of equipment utilizations are calculated and labeled the deviations for each equipment type. The equipment type having the largest deviation will relinquish some of its assignments to the equipment type having the smallest deviations. The procedure continues until all deviations are minimized.

A second improvement in the solution is sought by comparing the initially move cost with the allocated cost for the best existing solution. For all moves whose allocated cost is greater than the original move cost, all feasible equipment types are temporarily assigned and the total cost is recalculated. If a reduction in the total cost is realized, the temporary assignment is made permanent. The process continues until no reductions can be identified.
COFAD compares the results of the completed iteration with the prior iteration. If the cost of the material handling system and the number of changes in material handling equipment assignments varies by less than an initially input steady-state percentage, no further iterations are performed to improve either the layout or the material handling system.

Figure 2.7. An Example of a Layout Generated via COFAD (Tompkins and Moore 1980).

2.4.3 PLANET

PLANET, Plant Layout ANalysis and Evaluation Technique, has the same basic input requirements as CRAFT and may be utilized to generate and evaluate layouts for similar type problems. PLANET proceeds through three phases to generate a layout. The first phase involves the translation of the input data into a format which may be utilized by the algorithm. The second phase involves the selection of the order in which the departments are to enter the layout. The third phase involves the determination of the placement of the departments when they enter the layout. PLANET required input data which describe the departments and the material flows. There are three alternative methods of inputting the required specifying material flow data.
The first method of inputting the material flow data is to specify the production sequence by department for each part to be handled within the facility. In addition to material flow data, the cost per move per 100 ft is required for each part number. The costs, like the move costs in CRAFT, typically may not be determined and are often input as unity. PLANET converts the production data into a from-to chart and then, by adding the flow volumes in both directions of the from-to chart, PLANET develops a flow-between-cost chart which is utilized by the PLANET construction algorithms.

The second method of inputting the material flow data is by inputting a from-to chart directly. The from-to chart is once again converted to a flow between cost chart by PLANET.

The last method of inputting material flow data is the penalty matrix. The higher the penalty input between two departments, the more important is the closeness of these two departments. The matrix can be used to indicate the relative frequency and difficulty of moving materials between departments or to indicate the relationship data specified on a relationship chart. The penalty matrix is converted to a flow between cost chart by PLANET.

The flow between cost chart and placement priorities are the basic for the PLANET selection algorithms. The highest placement priority is 1 and the lowest is 9. Departments are placed in the layout by priority class. There are three selection algorithms within PLANET that are used to determine the order of placement of departments from the same priority class. The selection algorithms are labelled selection methods A, B, and C.

Selection method A chooses the departments to enter the layout based on flow-between costs. The first pair of departments to enter the layout must be in the highest priority group and have the highest flow-between cost. The next department to
enter the layout from among the unselected departments is in the highest priority group of unselected departments and has the highest flow-between cost with any department already placed in the layout. This procedure is continued until all departments have entered the layout.

Selection method B selects the first pair of departments to enter the layout in the same manner as selection method A. The next department to enter the layout from among the unselected departments is in the highest priority group and has the highest sum of flow between costs with all selected departments (i.e., those that have been placed in the layout). This procedure continues until all departments have entered the layout.

The first department to enter the layout in selection method C is the department in the highest priority group that has the highest sum of flow-between costs with all other departments. The next department to enter the layout from among the unselected departments is in the highest priority group departments. This is continued until all departments have entered the layout.

The PLANET placement routine begins by selecting the first two departments to enter the layout and placing them adjacent to one another in the center of the layout. Each additional department to enter the layout is positioned so as to minimize the increase in handling cost. The location that minimizes the increase in handling cost is determined using a trial-and-error procedure. The centroids of the departments in the layout and distances between the centroids and each point around the perimeter of the existing layout are first determined. By calculating the product of the distances and the flow-between cost chart, the volume-distance product is obtained and the minimum cost point is selected as the location about which the department is to be positioned. The procedure is then repeated for each department entering the layout.
2.4.4 CORELAP

CORELAP, C0mputerized RElationship LAYout Planning, constructs a layout for a facility by calculating the total closeness rating (TCR) for each department where the TCR is the sum of the numerical values assigned to the closeness relationships (A = 6, E = 5, I = 4, 0 = 3, U = 2, X = 1) between a department and all other departments. The department having the highest TCR is then placed in the center of the layout. If there is a tie for the highest TCR, the following tie-breaking rule is applied: the department having the largest area is placed. Next, the relationship chart is scanned and if a department is found having an “A” relationship with the selected department, it is brought into the layout. If none exists, the relationship chart is scanned for an “E” relationship, then an “I”, and so on. If two or more departments are found having the same relationship with the selected department, the department having the highest TCR
is selected; if a tie still exists, the tie-breaking rule is utilized. The third department to enter the layout is determined by scanning the relationship chart to see if an unassigned department exists that has an “A” relationship with the first department selected. If so, this department is brought into the layout. If a tie exists, the TCR and then the tie-breaking rule are utilized. If no unassigned department exists that has an “A” relationship with the second department, the procedure is repeated considering “E” relationships, then “I” relationships, and so on. If a tie occurs, the TCR and then the tie-breaking hierarchy are utilized. The same procedure is repeated for the fourth department to enter the layout, except the three departments previously selected are included in the search. The procedure continues until all departments have been selected to enter the layout.

Once a department is selected to enter the layout, a placement decision must be made. The placement decision is made by calculating the placing rating for the available locations of the department, where the placing rating is the sum of the weighted closeness ratings between the department to enter the layout and its neighbours.

\[ \begin{array}{ccccccc}
0 & 14 & 11 & 11 & 13 & 0 & 0 \\
0 & 14 & 12 & 16 & 16 & 0 & 0 \\
0 & 0 & 17 & 17 & 15 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array} \]

Figure 2.9. An Example of a Layout Generated via CORELAP (Tompkins and Moore 1980).

After the final CORELAP layout has been prepared, CORELAP evaluates the layout by calculating the layout score. The layout score is defined as:
2.4.5 ALDEP

ALDEP, Automated Layout DEsign Program, has the same basic data input requirements and objectives as CORELAP. The basic procedural difference between CORELAP and ALDEP is that CORELAP selects the first department to enter the layout and breaks ties with the total closeness rating, where ALDEP selects the first department and breaks ties randomly. The difference between CORELAP and ALDEP is that CORELAP attempts to produce the one best layout, whereas ALDEP produces many layouts, rates each layout, and leaves the evaluation of the layouts to the facilities designer.

As was previously mentioned, the first department ALDEP selects to enter the layout is selected randomly. The relationship chart is then scanned to determine if there is a department having an “A” relationship with the randomly selected first department. If one exists, it is selected to enter the layout. If more than one exists, one is randomly selected to enter the layout. If no departments have a relationship at least equal to the minimum acceptable closeness rating specified by the user, the second department to enter the layout will be selected randomly. Once the second department to enter the layout is selected, the selection procedure is repeated for the second department selected and all unselected departments. Once the third department is selected, the next department to enter the layout is determined by repeating the selection procedure. This process continues until all departments have been selected to enter the layout.

The placement routine within ALDEP begins by placing the first department in the upper left corner of the layout and extends it downward. The width of the downward extension of the department entering the layout is input by the user and is
termed the sweep width. The department is placed in the layout using the sweep pattern illustrated in Figure 2.5. Each additional department added to the layout begins where the previous department ends and continues to follow the serpentine path. When all departments have entered the layout, ALDEP rates the layout by assigning values to the relationships among adjacent departments. If a department is adjacent to a department with which it has an "A" relationship, a value of 64 is added to the rating of the layout. An "E" relationship adds 16, "I" adds 4, and an "O" relationship adds 1 to the rating of the layout. A "U" relationship has no effect on the rating of the layout and if two adjacent departments have all "X" relationship, 1024 is subtracted from the rating of the layout. ALDEP prints the layout and the rating and then returns to randomly, generate the first department to be selected for the next layout. The entire procedure is repeated. ALDEP can be used in this manner to generate up to 20 layouts and ratings per run. ALDEP will not print layouts whose rating is less than an initially input minimal score.

Figure 2.10. Overall Sweep Pattern Followed by ALDEP (Tompkins and Moore 1980).
2.5 Software Development Process

The software development process may be viewed as a spiral, as illustrated in Figure 2.12. Initially, system engineering defines the role of software and leads to software requirements analysis, where the information domain, function, behavior, performance, constraints, and validation criteria for the software are established. Moving inward along the spiral to design and finally to coding.

To test computer software, spiral out along streamlines of Figure 2.12. Unit testing begins at the vortex of the spiral and concentrates on each unit of the software as implements in the source code. Moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Taking another turn outward to validation testing, where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, arrive at system testing, where the software and other system elements are tested as a whole.
Considering the process from a procedural point of view, testing within the context of software engineering is actually a series of three steps that are implemented sequentially. The steps are shown in Figure 2.13. Initially, tests focus on each module individually, ensuring that it functions properly as a unit named unit testing. Unit testing makes heavy use of white box testing techniques, exercising specific paths in a module’s control structure to ensure complete coverage and maximum error detection. Next modules must be assembled or integrated to form the complete software package.
Integration testing addresses the issues associated with the dual problems of verification and program construction. Black box test case design techniques are the most prevalent during integration, although a limited amount of white box testing may be used to ensure coverage of major control paths. After the software has been integrated, a set of high-order tests is conducted. Validation criteria, established during requirements analysis, must be tested. Validation testing provides the final assurance that software meets all functional, behavioral, and performance requirements. Black box testing techniques are used exclusively during validation. After validated, system must be combined with other system elements (e.g., hardware, people, databases). System testing verifies that all elements mesh properly and that overall system function/performance is achieved.

Unit Testing

Unit testing focuses verification effort on the smallest unit of software design—the module. Using the detail design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors detected as a result is limited by the constrained scope established for unit testing. The unit test is always white box-oriented, and the step can be conducted in parallel for multiple modules.

Unit testing is normally considered an adjunct to the coding step. After source level code has been developed, reviewed, and verified for correct syntax, unit test case design begins. A review of design information provides guidance for establishing text cases that are likely to uncover errors in each of the categories discussed above. Each test case should be coupled with a set of expected results.
Unit testing is simplified when a module with high cohesion is designed. When only one function is addressed by a module, the number of test cases is reduced and errors can be more easily predicted and uncovered.

Integration Testing

Data can be lost across an interface; one module can have an inadvertent, adverse affect on another; subfunctions, when combined, may not produce the desired major function; individually acceptable imprecision may be magnified to unacceptable levels; global data structures can present problems.

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit-tested modules and build a program structure that has been dictated by design.

There is often a tendency to attempt nonincremental integration. All modules are combined in advance. The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because the isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop.

Incremental integration is the antithesis of nonincremental integration. The program is constructed and tested in small segments, where errors and easier to isolate and correct; interfaces are more likely to be tested completely, and a systematic test approach may be applied.

Validation Testing

At the culmination of integration testing, software is completely assembled as a package. Interfacing errors have been uncovered and corrected, and a final series of software tests – validation testing – may begin. Validation can be defined in many
ways, but a simple definition is that validation succeeds when the software functions in a manner that can be reasonably expected by the customer.

System Testing

Ultimately, software is only one element of a larger computer-based system (e.g., new hardware, and information), and a series of system integration and validation tests are conducted. However, steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.

The primary objective for test case design is to derive a set of tests that has the highest likelihood for uncovering defects in the software. To accomplish this objective, two different categories of test case design techniques are used: white box testing and black box testing.

White box tests focus on the program control structure. Test cases are derived to ensure that all statements in the program have been executed at least once during testing and that all logical conditions have been exercised. Basis path testing, a white box technique, makes use of program graphs (or graph matrices) to derive the set of linearly independent tests that will ensure coverage. Condition and data flow testing further exercise program login, and loop testing complements other white box techniques by providing a procedure for exercising loops of varying degrees of complexity. Hetzel describes white box testing as "test in the small" and black box testing as "testing in large". Figure 2.14. shows the geographical of white box testing.

Black box tests are designed to validate functional requirements without regard to the internal workings of a program. Black box testing techniques focus on the information domain of the software, deriving test cases by partitioning input an output in a manner that provides thorough test coverage. Equivalence partitioning divides the input domain into classes of data that are likely to exercise specific software function.
There is more than one method of accomplishing the transaction from input to output. The solution to the problem is visualized as a black box of unknown, unspecified content, having input and output as shown in Figure 2.15.

\[ X = A + B \]

Figure 2.14. White Box Testing.

Figure 2.15. Black Box Testing.
III. SYSTEM DEVELOPMENT

3.1 Overview

The COMPLAN prototype is developed based on the Simplified SLP as stated on the previous section, which solves layout-planning problems in six steps. Starting with Chart the relationships, Establish space requirements, Diagram activity relationships, Draw space relationship layouts, and Evaluate alternative arrangements. COMPLAN finished function in the step five of Simplified SLP, Evaluate alternative arrangements. For the last step, Detail the selected layout plan, planner will select the result from step five first and designers should detail it themselves.

This program attempts to develop a layout that maximizes the interdepartmental relationship where the interdepartmental relationship is defined in relationship chart. COMPLAN required the interdepartmental relationship and space requirement to be the input of program. The numerical value of interdepartmental relationship is required as optional due to these values are set as the standard of Simplified SLP procedure. However, the designer can set to new value to get the various results of layout.

COMPLAN uses the technique of probabilistic approached which is integrated from mathematical procedures and heuristics to solve the layout problem. In construction procedure, COMPLAN arranges the activity area in queue and place in the diagram one by one. There are “m” locations which close to the existing activity areas are available to the next incoming department. And COMPLAN selects the location that can minimize the whole distance and maximize the departmental relationship for the coming department.

Improvement algorithm is use to modify the initial diagram to optimize the departmental relationship. COMPLAN use this algorithm after the draft layout is generated. The department in draft layout will be moved to the available location or
interchange with other department. On which movement can minimize the whole distance and maximize the departmental relationship is made to the diagram. The procedure continues until no movement can be found that minimize the whole distance and maximize the departmental relationship, the search is then terminated.

3.2 COMPLAN Flowchart

As in previous mention, COMPLAN is developed by using the step of Simplified SLP as a process guide. Table 3.1 compares between COMPLAN process with Simplified SLP step.

Table 3.1. The Comparison of COMPLAN Process and Simplified SLP Step.

<table>
<thead>
<tr>
<th>COMPLAN</th>
<th>Simplified SLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Input activity area and its requirement area.</td>
<td>Step 1. Chart the relationships</td>
</tr>
<tr>
<td></td>
<td>Step 2. Establish space requirements</td>
</tr>
<tr>
<td>(2) Calculate TCR.</td>
<td></td>
</tr>
<tr>
<td>(3) Queue activity area.</td>
<td></td>
</tr>
<tr>
<td>(4) Locate activity area.</td>
<td>Step 3. Diagram activity relationships</td>
</tr>
<tr>
<td>(5) Improve layout diagram.</td>
<td></td>
</tr>
<tr>
<td>(7) Calculate evaluated score.</td>
<td>Step 5. Evaluate alternative arrangements</td>
</tr>
<tr>
<td>None</td>
<td>Step 6. Detail the selected layout plan</td>
</tr>
</tbody>
</table>
Figure 3.1. Overall COMPLAN Flowchart.
Figure 3.1 shows overall COMPLAN flowchart. The left side is the procedure and process of COMPLAN and the right side is the result that comes out from individual procedure or process. COMPLAN process starts with:

1. Input activity area and its requirement area.
2. Calculate TCR.
3. Queue activity area.
4. Locate activity area.
5. Improve layout diagram.
7. Calculate evaluated score.

According to COMPLAN flowchart, there are 7 processes in COMPLAN to be executed. The individual process has its own function and passes its result to the next process.

1. Input activity area and its requirement area.

This is the first step of COMPLAN to input all requirement data before generating layout. The activity area’s name, space requirement of each activity and relationship diagram area are input to the program. COMPLAN will keep this information until last step is processed. Figure 3.2 had shown the example of inputting requirement data. And Figure 3.3 had shown the relationship diagram of COMPLAN.
Figure 3.2. An Example of Inputting Step of COMPLAN.

Figure 3.3. The Relationship Diagram of COMPLAN.
(2) Calculate TCR.

This is the second process in COMPLAN. This process is used to calculate TCR to weight on all activity area. This weight will be useful for the next step of COMPLAN. The higher weight activity area will be treated as higher priority. The detail of this algorithm is stated in the layouts development algorithm part. The flowchart of calculate TCR step is shown in Figure 3.4.

When COMPLAN receive the corrected data, COMPLAN converts the departmental relationships into the numerical weight and gather up for each activity area. Normally “A” rating equals 4 point, “E” rating equals 3 point and 1 point deduct until “X” rating equals –1 point. COMPLAN sums up all point in each activity area. The first is to select the first activity area from list and look for interdepartmental relationship, sum up the relationship one by one until the last relationship is completed. Then select the next activity area from the list and repeat this step until the last activity area is done. The result is the TCR score of each activity area. In the Figure 3.5, TCR score of each activity are shown in “SUM” column.
Figure 3.4. Calculate TCR Procedure.
(3) Queue activity area.

After summing up TCR score of all activity areas, the program can select the first entering activity area and choose other activity area that is most related to the previous selected activity area until all activity areas are selected. The activity area that has the highest TCR score is selected as the first entering activity area. All activity areas are arranged in queue. The process is illustrated in Figure 3.6.
SELECT HIGHEST TCR ACTIVITY AREA

SELECT HIGHEST RELATIONSHIP WITH ANY PREVIOUS ACTIVITY AREA

ALL ACTIVITY AREA ARE SELECTED

Yes

No

Figure 3.6. Queue Activity Area Procedure.

(4) Locate activity area.

When COMPLAN completed arranging the activity area in queue then COMPLAN locates the activity area from queue which is created in queuing process into the block diagram by using steps in the Figure 3.7. The first department in queue is located at the middle of diagram and the other is located as close as possible to the existing activity area. One unit block is expressed to one activity area. This step results one unit block layout as the Figure 3.8.
Figure 3.7. Locate Activity Area Procedure.
(5) Improve layout diagram.

This step examines all departmental relationships in block layout for which one is unsatisfied and tries to move to another location without breaking other relationships. By using mathematics method, the overall relationship score is re-calculated in every movement. If the relationship score of that movement is higher than the existing one, that movement is successful. This step is running as recursive process until no movement has the higher relationship score. More detail about this process is stated in layouts improvement algorithm part and Figure 3.9. shows improvement process flowchart.
SEARCH FOR NON-ADJACENCY RELATIONSHIP

SELECT UNSATISFY ACTIVITY AREA

SELECT POSSIBLE LOCATION

CALCULATE DIAGRAM SCORE FOR NEW LOCATION

ALL POSSIBLE LOCATION ARE CALCULATED

ALL ACTIVITY AREA ARE CALCULATED

ANY NEW GREATER LAYOUT DIAGRAM SCORE

INTERCHANGE LOCATION REFER TO NEW LAYOUT DIAGRAM SCORE

Figure 3.9. Improve Layout Diagram Procedure.
(6) Draw layout.

The first step of this process is started from searching the separated grid in block diagram. This process search on both vertical and horizontal grid. The grid which has the least space requirement ratio between both sides is selected, such as the horizontal grid between area 6, 3, 1, 7, 2, 4 and 9, 10, 5, 8 of locating activity area step equal 23 : 32. The layout diagram is also horizontally separated in that ratio. Next the first separated side is selected to do this process again until only one activity area is left; then process on another separated side. Figure 3.10. shows the process flowchart of this step.

After improvement layout diagram of the activity area, COMPLAN will draw the layout of activity area with their relative space requirement according to the layout diagram position. From the same example above, the layout can be generated as the Figure 3.11. In the one unit block layout, the first row is located by activity area 6, 3 and 1 respectively. In layout diagram, these activity areas are located at the same position with its required space.
Figure 3.10. Draw Layout Procedure.
(7) Calculate evaluated score.

COMPLAN also provides the evaluated score of each generated layout for planner in selecting the satisfied layout. This evaluation is based on mathematics method, the satisfied departmental relationship is calculated as a plus score and unsatisfied departmental relationship is calculated as a zero score. More satisfied relationships become higher evaluated score. Figure 3.12 is an example of evaluated score, the total score is summing up in “total” row.

Figure 3.13 shows steps of evaluation process which start with searching for the adjacency in the layout diagram. The adjacency is counted as a positive score and non-adjacency is counted as zero score. Then all adjacency will multiple with its value rating and sum up as the evaluation score. More detail is stated in layouts evaluation part.
Figure 3.12. An Evaluation Score in COMPLAN.

<table>
<thead>
<tr>
<th></th>
<th>Alternate 1</th>
<th>Alternate 2</th>
<th>Alternate 3</th>
<th>Alternate 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (4)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>E (3)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>I (2)</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>O (1)</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>U (0)</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>X (-1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>32</td>
<td>31</td>
<td>33</td>
<td>30</td>
</tr>
</tbody>
</table>

3.3 Layouts Development Algorithm

Thinking of the most relative department should be located at the center of layout, Total Closeness Rating (TCR) is the concept to be considered for determining the first department to place in the layout. The departmental relationships are considered as quantitative scale such as “A” should be 4, “E” should be 3, and etc. Then continue with calculating TCR, TCR is computed by summing up all closeness ratings associated a particular department. The formula for calculating the TCR of a department is stated as follows:

$$TCR = \sum_{i=1}^{n} \sum_{j=1}^{n} rel_{ij}$$

where $rel_{ij}$ is the closeness rating between department i and department j.

$n$ is the total number of departments.
Figure 3.13. Calculate Evaluated Score.
After all departments are calculated TCR, the department with the highest TCR will be selected as an initial department for placement in the layout. The next department which has the highest closeness rating with the first is chosen next and placed adjacent to the previous area. The following departments will then be selected based on its highest rating to any former departments. The procedure continues until all departments are laid out on the plan. The result is the sequence of the department number which was prompted to be placed on layout.

Table 3.2. Example Problem I.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>O</th>
<th>U</th>
<th>X</th>
<th>TCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O</td>
<td>E</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>I</td>
<td>O</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>O</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>I</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>O</td>
<td>U</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>U</td>
<td>E</td>
<td>U</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>U</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>A</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>I</td>
<td>E</td>
<td>U</td>
<td>E</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>I</td>
<td>A</td>
<td>U</td>
<td>U</td>
<td>A</td>
<td>E</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

From the table above, the “A”, “E”, “I”, “O”, “U”, and “X” rating are considered as value 4, 3, 2, 1, 0, and -1 respectively. There are two highest TCR, the first met department is selected. Then the first selected department is department no. 2. The next selected department is the department that has the highest closeness rating with department no. 2. There are also six departments that has an “I” rating, department no. 3, no. 5, no. 6, no. 7, no. 9 and department no. 10. Department no. 3 is the first met department then department no. 3 is selected. The next department that has the highest closeness rating with any former department no. 2 or department no. 3, will be selected
next. It is department no. 1, has an “E” with department no. 3. The selection routine continues until no department is remaining. The result might be 2, 3, 1, 5, 10, 8, 9, 7, 6 and 4.

COMPLAN will use this sequence in locating department into the relationship diagram. The first department is placed in the middle of the relationship diagram. The next department is located where it has the lowest sum of multiple distances and the closeness rating between itself and all previous departments, as expressed below:

\[ \text{Min} \sum_{i=1}^{n} \sum_{k=1}^{n} \text{rel}_{ik} \text{dis}_{ik} \]

where \( \text{rel}_{ik} \) is the closeness rating between the activity area i and k.
\( \text{dis}_{ik} \) is the distance between the activity area i and k.
\( i \) is the selecting order of activity area.
\( k \) is the incoming activity area.
\( n \) is the number of activity areas being considered.

The distance required in the previous equation can be calculated using Euclidean distance which represents a linear distance between two centroid locations. The formula for determining the Euclidean distance is as follows:

\[ \text{Distance} = \sqrt{\left( X_i - X_j \right)^2 + \left( Y_i - Y_j \right)^2} \]

where \((X_i,Y_i)\) and \((X_j,Y_j)\) represent the coordinates of locations i and j, respectively.

When tied in selecting location, the location which has the minimum distance between its own centroid with the exiting location centroid will be selected. With the same departmental relationships of the problem I and fixed the sequence of locating is 10, 8, 5, and etc. The following figure can be explained the COMPLAN procedure in selecting location when tie situation.
Figure 3.14. A Locating Method by COMPLAN.

In Figure 3.14(a), the first department no. 10 is located in the middle and the location available for the next department are location A, B, C, and D. COMPLAN searches all the available locations from top left to bottom right for the next coming department, no 8. All possible locations have the same distance between centroid, one unit length, and value of \((\text{Rel}_{ij} \times \text{Dis}_{ij})\), 4 multiply by 1, then the first met location, location A, is selected as shown in Figure 3.14(b). Then the available locations are changed to be E, F, G, H, I, and J. COMPLAN finds the minimum value of \((\text{Rel}_{ij} \times \text{Dis}_{ij})\) department no. 5 related to department 10 and 8 at location H, I and J. The value is 4 which came from 4 multiply by 1 for department 10 plus 0 multiply by 1.414 for department 8 or expressed in equation of \((4 \times 1)_{10} + (0 \times 1.414)_{8}\). The minimum centroid distance is only at location H and I. Then the first met location, location H, is selected as shown in Figure 3.14(c).

3.4 Layouts Improvement Algorithm

The objective is to maximize the adjacency. In the relationship diagram, any department that lacks the relationship adjacency with other departments are candidates for improvement. These departments will be placed in various positions to find the highest relationship score. The relationship score is computed by using the following equation:
The relationship score = \( \sum_{i=1}^{n} \sum_{j=1}^{n} rel_{ij} \cdot adj_{ij} \cdot eff_{ij} \)

where

- \( rel_{ij} \) is the user-assigned closeness rating between activity areas \( i \) and \( j \).
- \( adj_{ij} \) is the adjacency between activity areas \( i \) and \( j \).
  
  \( 0 \) : non-adjacency
  
  \( 1 \) : adjacency

- \( eff_{ij} \) is the percent efficiency of adjacency between activity areas \( i \) and \( j \), calculated from the reciprocal of the distance between two locations.
  
  \( 0.7 \) : Corner adjacency (1 divided by 1.414 unit length)
  
  \( 1 \) : Side adjacency (1 divided by 1 unit length)

\( n \) is the number of departments being considered.

In this interchange routine, which interchange layout offers the highest relationship score will be selected. The interchange routine will process until no higher relationship score is met.

3.5 Layouts Evaluation

Layout scores are computed from alternative layouts developed in the previous steps. Satisfactory relationships between activity areas will be converted to scores. Only satisfactory adjacency will be counted as an absolute positive score.

\[ \text{Layout scores} = \sum_{i=1}^{n} \sum_{j=1}^{n} rel_{ij} \cdot adj_{ij} \]

where

- \( rel_{ij} \) is the standard closeness value between activity areas \( i \) and \( j \).
- \( adj_{ij} \) is the adjacency between activity areas \( i \) and \( j \).
  
  \( 0 \) : non-adjacency
  
  \( 1 \) : adjacency

\( n \) is the number of activity areas being evaluated.
By following the Simplify SLP method, the "A", "E", "I", "O", "U", and "X" rating is counted as 4, 3, 2, 1, 0, and -1 respectively.

3.6 Data Requirements

The data requirements for COMPLAN include:

1. The number and names of activity areas.
2. Each activity area is expressed in square units.
3. Departmental closeness ratings.
4. In case of the improvement method, an initial layout is also required as an input.

3.7 Peripheral Requirements

The graphical computerize layout planning software requires the following hardware and software to operate.

1. Hardware must be at least Pentium Processor, 32 MB of RAM, 3 MB of space, disk drive 3.5", color monitor, mouse and printer.
2. Operating system must be at least Microsoft Windows 95.

This is the minimum requirement of this graphical computerized layout planning. The higher hardware specification specially microprocessor and memory will increase software performance.

3.8 Graphical User Interface

This software is developed by using Object Oriented Programming (OOP) language named Delphi, which is operated on Windows-based systems. Delphi is a Pascal compiler and editor with a lot of tools for visual programming. The visual programming allows designer simultaneous designing and writing of the program. The primary advantage of a visual programming language is its excellent graphical user
interface. This enhances the program’s user friendliness. The Figure 3.15 shown the first page of Delphi 5.0.

Figure 3.15. Illustration of Delphi 5.0.
IV. SYSTEM EVALUATION

4.1 Overview

Software testing is one element that is often referred to as verification and validation (V&V). Verification refers to the set of activities that ensure that software correctly implements a specific function. Validation refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements. Boehm states this another way:

Verification: “Are we building the product right?”

Validation: “Are we building the right product?”

Glen Myers states a number of rules that can serve well as testing objectives:

(1) Testing is a process of executing a program with the intent of finding an error.

(2) A good test cast is one that has a high probability of finding an as yet undiscovered error.

(3) A successful test is one that uncovers an as yet undiscovered error.

The above objectives imply a dramatic change in viewpoint. They move counter to the commonly held view that a successful one in which no errors are found. Our objective is to design tests that systematically uncover different classes of errors and to do so with a minimum amount of time and effort.

If testing is conducted successfully (according to the objective stated above), it will uncover errors in the software. As a secondary benefit, testing demonstrates that software functions appear to be working according to specification, that performance requirements appear to have been met. In addition, data collected as testing is conducted provide a good indication of software reliability and some indication of software quality as a whole.
Information flow for testing follows the pattern described in Figure 4.1. Two classes of input are provided to the test process:

1. A software configuration that includes a software requirements specification, a design specification, and source code.

2. A test configuration that includes a test plan and procedure, any testing tools that are to be used, and test cases and their expected results.

Actually, the test configuration is a subset of the software configuration.

![Test Information Flow](image)

Tests are conducted and all results are evaluated. That is, test results are compared with the expected results. When erroneous data are uncovered, an error is implied and debugging commences.

### 4.2 Verification

Verification is generally done in the development environment such as check the system operation, user interface, and error free. On the development process, this prototype program is separated written following 5 steps of simplified SLP which is shown in the previous flow chart. Each step is broken down into several modules and each module is verified with both unit testing and integration testing.
Test is designed to uncover errors in the following categories:

(1) Improper or inconsistent typing.
(2) Erroneous initialization or default values.
(3) Inconsistent data type.
(4) Processing incorrect.

As described in the previous part, unit testing is normally considered an adjunct to the coding step. This testing method, the developed program is debugged by Delphi program itself by tracking the data transaction and the result step by step to ensure all function works properly as coding. If the problem occurs, the program will be re-coded and re-tested again. In case of no problem found, the internal coding verification is finished. Then another method will be used to verify again by inputting several preparing data into module and compare the result with the manual made. If nothing different, unit testing is finished for this module. All modules and functions should pass this testing before the integration testing.

Integration testing is made when every module is finished unit testing. One module links to another module and run testing again. According to unit testing verifies the function in each module work properly, then this testing tests only the transaction between each module is right and all data are passed correctly. All modules are gathered up one by one and tested with the same data. Tracking the result between each module and the whole set, compared with the manual made. If the results are the same, it can be considered that the internal transaction has no error.

The other part for this program is user-interface part. Normally some of the errors of program comes from the user. The user may key in the wrong or undefined type of data to program so it is necessary to prevent this activity. The user must input the corrected data into program and program will process next. This program has the data
integrity system to verify all input data for validity before passing to process. This system will display the fault when the incorrect data is inputted. As describe in the data requirement section, the raw data is restricted by their proper data type.

The input data type

Name: The input should be any alphabet and no limitation. This input is only used for remind the user to identify the department’s name.

Area: This variable is the most important and has high possibility of inputting error such as using alphabet instead of using numerical data. To prevent abnormal input, the program allows only positive number between 1 to 9999 to be input. It is restricted by the input digit is not greater than 4 digits. This space requirement area is normally used to be calculated in many functions. By restrict this type of this input can prevent some error in calculation function. Figure 4.2. is shown when the input number is zero or no input number.

Figure 4.2. Error Message.
Relationship: This variable is also important as Area variable. This variable will effect directly with the result of layout. The departmental relationship only A, E, I, O, U and X is acceptable due to only these six symbols have meaning in layout planning procedure. In Simplified SLP, these vowels and X are used as the symbol of departmental relationship, other letter has no meaning for the program then program allows only these 6 letters to be process.

4.3 Validation

As previously mentioned, this method is used to assure this software meet the customers' requirements. The major point is this software should generate the possibly result of layout planning. Two methods are proposed for program validation. These are compare the result with other computerized layout planning software and compare the result with expert solution.

(1) Compare the result with other computerized layout planning software.

This comparing is subjected to determine that this developed software performance has met the same level with other computerized layout planning software. The departmental relationship of the recommended result is considered as the first priority and other criterias are also considered next, such as user-friendliness, graphics and time consuming. In this comparing, starting with entering the same input data into both computerized layout planning program, and operating them until the recommended results are obtained from both programs. Then, the interested criterias are compared to other programs.

The computerized layout planning software was developed since 1964 and is continued developing until now. For this comparing, the
selecting program to be compared with COMPLAN is BLOCPLAN-90 because BLOCPLAN-90 is a facility layout system that has been developed at the Industrial Engineering Department of the University of Houston and is continued to be developed. The other reason is the objective of development BLOCPLAN-90 is for using technique in design layout planning, not for the commercial use and it also framed in a few group of planning users.

These are the factors considered:

(a) The result of computerized layout planning. Program should analyze the departmental relationship and process through all input data. The layout result should be practical and patterned in a realistic shape. The departmental relationship and space requirement should be related with the input data.

(b) The capability of solving large size problems. It is determined by the number of activity area that the program can support. The large number of activity area made the program more complicated calculation due to the large number of activity area will make a larger number of possible result.

(c) Time consuming. Some software can generate the most effective result but it takes much time. It is not practical that this factor is considered. The average time consumed in calculating and designing process until generating the layout after all required data input are considered. This factor is a qualitative factor because it cannot be measured in accurate time. This factor is also directly concerned to computer specification then it is easy to dispose as fast and slow.
(d) User friendliness. The ease of use of the program since input process until output process including editing data are considered. The summary of comparing result is in Table 4.1:

Table 4.1. Comparison between BLOCPLAN-90 and COMPLAN.

<table>
<thead>
<tr>
<th>Consideration factor</th>
<th>BLOCPLAN</th>
<th>COMPLAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>The capability of solving problem</td>
<td>- The maximum activity area is 18 area. - Maximum 20 alternatives.</td>
<td>- The maximum activity area is 20 area. - Unlimited alternatives.</td>
</tr>
<tr>
<td>Time consuming</td>
<td>Treat as normal</td>
<td>Faster: skip viewing interchange location. Slower: view interchange location.</td>
</tr>
<tr>
<td>User friendliness</td>
<td>A lot of manus choice. User must understand the menu first.</td>
<td>Very ease of use by following the program steps. All data sheet follows the actual work.</td>
</tr>
<tr>
<td>Layout pattern</td>
<td>Patterned in the block shape with layout score.</td>
<td>Pattern in the color block shape with layout score. User can compare the four results in the same time and choose the preferred one.</td>
</tr>
<tr>
<td>Others</td>
<td>- Operate in DOS base. - Can solve multistory problem. - Varied in output layout shape. - Consider flow of material as input. - Difficult to edit data</td>
<td>- Operate in Windows base. - Can't solve multistory problem. - Fixed output layout in square shape. - Not consider flow of material. - Print out the result with its dimension and relation chart. - View layout in larger size. - Ease of edit data.</td>
</tr>
</tbody>
</table>
(2) Compare the result with expert solutions.

This is the second method of validation. Through recommended layout plans provided by experts, compares the results developed by the system and those done by the experts. To compare the results between these two programs, COMPLAN and BLOCPLAN-90, and the expert result. To prevent error from the controllable factor, these two programs are set in the value of “A”, “E”, “I”, “O”, “U”, and “X” equal 10000, 1000, 100, 10, 1 and −10000 respectively in calculating TCR and generating layout pattern. And using value of 4, 3, 2, 1, 0, and −1 for the “A”, “E”, “I”, “O”, “U”, and “X” closeness rating respectively in evaluating score process. The evaluated score is determined by summing up all adjacency closeness rating. The first testing problem is selected from the book of Richard L. Francis, Leon F. McGinnis, Jr., and John A. White as shown in Table 4.2.

Table 4.2. Example Problem II.

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>O</th>
<th>U</th>
<th>X</th>
<th>AREA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>E</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>1200</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>O</td>
<td>I</td>
<td>I</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>3</td>
<td>E</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>300</td>
</tr>
<tr>
<td>4</td>
<td>O</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>400</td>
</tr>
<tr>
<td>5</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>6</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>O</td>
<td>U</td>
<td>E</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>300</td>
</tr>
<tr>
<td>7</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>U</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>900</td>
</tr>
<tr>
<td>8</td>
<td>U</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>600</td>
</tr>
<tr>
<td>9</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>U</td>
<td>E</td>
<td>E</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>1000</td>
</tr>
<tr>
<td>10</td>
<td>U</td>
<td>I</td>
<td>U</td>
<td>I</td>
<td>A</td>
<td>U</td>
<td>U</td>
<td>A</td>
<td>E</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>3000</td>
</tr>
</tbody>
</table>
The results of problem in Table 4.2. are shown in Figures 4.3., 4.4. and 4.5. with evaluated score.

Figure 4.3. The Recommend Result via Expert with 29 Evaluated Score.

Figure 4.4. The Result via BLOCPLAN-90 with 33 Evaluated Score.
Figure 4.5. The Result via COMPLAN with 34 Evaluated Score.

The other testing problem is select from the book of Richard Muther and John D. Wheeler as shown in Table 4.3.

Table 4.3. Example Problem III

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>O</th>
<th>U</th>
<th>X</th>
<th>AREA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>O</td>
<td>A</td>
<td>O</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>120</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>O</td>
<td>I</td>
<td>E</td>
<td>O</td>
<td>O</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>4</td>
<td>O</td>
<td>X</td>
<td>I</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>5</td>
<td>I</td>
<td>I</td>
<td>E</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>U</td>
<td>U</td>
<td>O</td>
<td>U</td>
<td>U</td>
<td>E</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>7</td>
<td>U</td>
<td>U</td>
<td>O</td>
<td>E</td>
<td>U</td>
<td>E</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>80</td>
</tr>
</tbody>
</table>
The results of the problem in Table 4.3. are shown in Figures 4.6, 4.7 and 4.8 with evaluated score.

**Figure 4.6.** The Recommend Result via Expert with 18 Evaluated Score.

**Figure 4.7.** The Result via BLOCPLAN-90 with 21 Evaluated Score.
Figure 4.8. The Result via COMPLAN with 21 Evaluated Score.
V. CONCLUSIONS & RECOMMENDATIONS

5.1 Conclusions

After, the development and evaluation of layout planning were accomplished subjectively by planners who resorted to graphical techniques and template manipulation, and most layout planning problem is so complicated a problem that it also deals with a large amount of data source. Then many computer-aided layout programs are developed to analyze the based procedure in layout development since 1963 such as CRAFT, COFAD, PLANET, CORELAP and ALDEP. Each program uses different methods to generate the layout.

This thesis attempts to develop a new graphical computer-aided layout program named COMPLAN (Computerized layout planning). The purpose of developing this program is to integrate the new layout designing technique into computer software by emphasizing on the user-friendliness and time-saving.

By studying various programming languages, Delphi is selected to be used to develop this layout planning software named COMPLAN. Delphi is an Objected Oriented Programming (OOP) which is operated on Windows-based system. And it has ease of studying because the command is similar to English language.

Based on the procedures of this program, COMPLAN requires the basis data such as activity area’s name, space requirement and interdepartmental relationship as normal input as other programs. This prototype is developed for giving user ease of use in designing layout. It begins with the user interface, COMPLAN operated on windows base then this makes user more ease of use than DOS base. The next, Simplified SLP technique is selected to use in this prototype. This technique has 6 systematic steps in designing layout. Only first 5 steps are used in COMPLAN because the last step deals with internal layout. COMPLAN generates alternative layout for user to making
decision and then after user selects the final layout, user will detail internal layout themselves. The designer can operate COMPLAN by seeing the symbol of Simplified SLP or following the step number on that symbol.

In computerized program, all inputs should be done before the operation. Then the first step, Establish space requirement, is exchanged from the second step in Simplified SLP to be the first step in COMPLAN. In this step, the designer can input the name, space requirement of the activity area and also choose the symbol of its activity area. All input data can be edited all along the problem even if this step is passed, the updated data will effect the whole program.

The second step, or the first step in Simplified SLP, is named Chart the relationships. The advantage of this step is user interface. The input form is designed the same as the original one in Simplified SLP relationship chart. The designers who are familiar with those forms can use this program easily. This step will not use manual data entry, all uses computer's pointer. The initial interdepartmental relationships are all 'U' and the designer can change it by clicking on left-mouse button. The relation will be changed upward from 'X' to 'A' and go back to 'X' again. To prevent user confusing, the program shows the pair of activity areas with its interdepartmental relationship when user lays mouse on any the relationship. And also the user can verify the amount of each relationship in the small window before processing the next step.

In designing block diagram of activity relationship step, COMPLAN shows 4 alternative block diagrams with the diagram scores and time consumed. This score indicates on block diagram which is not considered on space requirement. This makes the designer to choose the most preferable one (if any) or they can re-design unlimited new alternate layouts or re-arrange some activity area of the existing one by
themselves. The other feature is improvement function. This function is used to improve the manual made block diagram to get higher score.

The fourth step, Draw space relationship layout, shows the layout diagram according to the previous block diagram and its space requirement. User can larger the layout by using right-mouse button and select zoom option. The zoom window enlarges the selected layout and indicates the dimension of each activity area. And user can save layout into picture by using right-mouse button and select save option.

The last step of COMPLAN, Evaluate alternative arrangement, shows score of each layout diagram for the user to make the decision on it. The higher score shows the better relationship. More features in the evaluating layout step, COMPLAN can show the source of layout score which is satisfied-unsatisfied interdepartmental relationship. Then the designer can look back to the designing block diagram step and re-arrange the significant interdepartmental relationship to get the higher layout score.

Based on the comparison with other similar programs and expert solution, COMPLAN looked a bit better in user interface, time consumption and the result of COMPLAN is better than the recommended expert solution and has no significant score with BLOCPLAN.

COMPLAN can generate many alternated layout results. For more results, user can customize the weight of the relationship value to get the various result of layout. Additionally, COMPLAN can also save and print out the result. The format of save file is bitmap or BMP. It can be viewed by several images viewer program such as imaging in Windows.
5.2 Recommendations

Based on the result of study, this program seemed to be as good as an expert in solving problems by considering evaluated score. COMPLAN can analyze the data and generate layout much more quickly than actual work, also the various results can be selected. COMPLAN should help in designing layout which is not larger than 20 activity areas. This number is fixed by the Simplified SLP technique. The results that come out are convincible because they are obtained by using Simplified SLP technique which is the famous technique at present. For more customized use, user can config the interdepartmental weight on each relationship. Then the various alternatives are generated subjected to customer's need. This program uses heuristic method to solve the problem then no correct answer is made. There are all possible answers. Users can use these alternatives as a guide in their work by looking at the diagram score and evaluated score. However after selecting the preferred layout, the designer usually modifies the layout solution obtained from COMPLAN to be more realistic and useful design because COMPLAN can only generate the layout solution in square shape. Users can save the result as picture and print out on the paper. Then they can review the problem anytime. In addition, this program does not produce a detailed layout in the last step of Simplified SLP. Hence, the output produced by this program should not be considered as the final step in designing process.

5.3 Future Research

The result of this program is achieved as stated in the previous section. This program might be useful for layout planner in designing layout. This program is the prototype of graphical computerized layout planning software then this program is suitable for solving the noncomplex problem. Because of the complex problem need more complicate process of designing layout and process of generating layout.
However, this program should be expanded for solving large-size problems with more complicated data, generating multistory layout and generating in customized layout shape for more realistic solution. For more complicated problems, other techniques and conditions are needed to consider such as material cost, material flow and transportation cost. To prevent the memory effect problem while running COMPLAN, this program can only generate unlimited alternative results but it cannot store all alternative results into a set of result. So COMPLAN cannot view the alternatives backwards. Only four alternatives are viewed. For more convenience to the user in making decision, this program should be viewed results backwards and forwards. The designer can trace all the possible results and its evaluation score to make decision. On the larger system, COMPLAN could be adapted as a client-server program for solving large-size problem. After selecting the preferable layout, each user could pick up any floor plan to design their own detail layout on AUTOCAD and save it back.
Introduction

COMPLAN was first developed in Computer and Engineering Management Faculty, Assumption University to provide user more convenience in designing layout plan. COMPLAN provides the solution for the general planning, and up to engineering level.

Featuring high user friendliness, user can use both keyboard and pointer to control, operate, and fit COMPLAN solution.

COMPLAN provides the solution by following the famous method of designing layout, Simplified Systematic Layout Planning (Simplified SLP) method that was created by Richard Muther.

COMPLAN provides the optimal layout solution based on the user closeness rating weight and calculate the overall score based on the standard closeness rating.
Operating COMPLAN

First running COMPLAN, the main window of COMPLAN is looked like:

![COMPLAN Main Window](image)

Figure A.1. Illustration of COMPLAN.

The left side is the number and name of designing step. There are five steps and the initial step is step number 1. The gray number means that step does not active. There are four menus in the top of COMPLAN, (File, View, Config and About). Only File menu and View menu have sub-menu.
File menu

There are seven sub-menus in this menu:

![File Menu Diagram](image)

Figure A.2. Illustration of File Menu.

1. New is used to clear all the existing problem and ready for new problem.
2. Open is used to load the prepared data.
3. Save is used to save the text data into a DAT file.
4. Print Setup is used to setup the printer.
5. Print is used to print the problem and solution. User can select which result will be printed out by checking in the check box.
6. Exit is used to close COMPLAN.
Figure A.3. Illustration of Print Menu.

View menu

This menu shows the steps that were already processed in COMPLAN. The gray menu means that step is not process yet. User can view any process by using this menu. Used to view the step of COMPLAN. It can be used hot key F1 to F6.

(1) F1 is Step 1 Establish space requirement
(2) F2 is Step 2 Chart the relations
(3) F3 is Step 3 Diagram activity relationships
(4) F4 is Step 4 Draw space relationship layout
(5) F5 is Step 5 Evaluate alternative arrangements
(6) F6 is TCR Table
<table>
<thead>
<tr>
<th>View</th>
<th>Config</th>
<th>About</th>
</tr>
</thead>
<tbody>
<tr>
<td>Establish space requirement</td>
<td>F1</td>
<td></td>
</tr>
<tr>
<td>Chart the relations</td>
<td>F2</td>
<td></td>
</tr>
<tr>
<td>Diagram activity relationships</td>
<td>F3</td>
<td></td>
</tr>
<tr>
<td>Draw space relationship layout</td>
<td>F4</td>
<td></td>
</tr>
<tr>
<td>Evaluate alternative arrangements</td>
<td>F5</td>
<td></td>
</tr>
<tr>
<td>TCR Table</td>
<td>F6</td>
<td></td>
</tr>
</tbody>
</table>

Figure A.4. Illustration of View Menu.

Figure A.5. Illustration of TCR Table.
There are three categories in this config windows there are value rating, generate method and improvement method. Only value rating and generate method can be configured.

(1) To set the user closeness rating weights and select generating solution method.

The default setting is 4, 3, 2, 1, 0, and -1 for A, E, I, O, U, and X rating respectively. The default method is generating by random activity sequence.

![Figure A.6. Illustration of Config Menu.](image)

The easy way to operate COMPLAN is only by following the step 1 to 5. The initial step is number 1, Establish space requirements. Enter the name of activity area or department in the named blank. And enter positive number which is not greater than four digits in the area required blank. Function blank is not essential in designing process but it will be shown in the printout solution paper.
And press Add button to enter the data into the below table as the following figure.

Figure A.7. Illustration of Step 1 Establish Space Requirements.

To insert data by first selecting the location in the table and then entering data in blank and press Insert button. The data will be inserted in that location.

To correct data by double click on the data that needed to be corrected. All the data is shown in the blank and ready to be corrected. When finish, press Update button.

To delete data by selecting the data that needed to be deleted in the table and press Delete button. The data will be deleted.
To clear all data by pressing Clear button. The confirmation box is shown and process will be done after yes confirmation.

When all data are already input, Step 2, Chart the relations, is turned to be ready to process by clicking the button on the left side. The relation chart is ready to be input. The initial relationship is “U” for all relation.

![Diagram of Step 2 Chart the Relations](image)

Figure A.8. Illustration of Step 2 Chart the Relations.

To input the departmental relationship by using mouse click on the relationship, it will be turned up from “U” to “O” until “A” and then turned to “X” again. The program also shows the position of departmental relationship at the top.
The buttons on the right side are used to indicate the requirement relationship to compare with the number of relationship in the bottom table.

Color check box is used to fill color in relationship. If it is unchecked, the relationship color is turned to be white otherwise it turns to be colored.

Rating check box is used to show the rating table in small window. If it is uncheckked, the rating table disappears. User can check the number of each relationships in the rating table.

Step 3, Diagram activity relationships, is used to generate layout diagram with alternates by clicking Gen button on the upper right side. Generated time and diagram score is shown after finish generated each diagram. The left side score is the percentage diagram score and the right side is the generated time on each diagram. The first is on the top-left, the second is on the top-right, the third is on the bottom-left and the forth is on the bottom-right.

Four buttons in Manual arrange group are represented to the locations of each diagram. If the button shows at “ON” position, the relative diagram can be modify or re-arrange by drag and drop the selected department. The diagram score will be changed simutaneous but the generated time will not be changed. To improve individual diagram by pressing Improve button and select the relative diagram button. Only relative diagram in “ON” position will be improved by Improve button.
For getting the solution faster, user can pass to step 4 after pressed Gen button without seeing diagram generating. The processing time of illustrate the graphical drawing take more time consuming.

Step 4, Draw space relationship layout, converts the solutions that were generated in step 3 to the block layout.

User can select the preferable one or return to step 3 to generate other alternate. On each layout, user can use right click button on mouse. Two menus are shown, Zoom picture and Save picture. Zoom picture is used to larger the current layout and save picture is used to save the current layout in bitmap format.
Figure A.10. Illustration of Step 4 Draw Space Relationship Layout.

Figure A.11. Two Additional Menu on Right Mouse.
### Step 5: Evaluate alternative arrangements

Evaluate alternative arrangements, shows the layout score with alternate. User also checks the adjacency between departments by pressing the check button under the evaluated table.

#### Illustration of Zoom Function

<table>
<thead>
<tr>
<th>Size</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: 18.13 x 5.52</td>
<td>100</td>
</tr>
<tr>
<td>2: 15.72 x 12.72</td>
<td>200</td>
</tr>
<tr>
<td>3: 16.55 x 18.13</td>
<td>300</td>
</tr>
<tr>
<td>4: 16.70 x 23.96</td>
<td>400</td>
</tr>
<tr>
<td>5: 18.14 x 27.57</td>
<td>500</td>
</tr>
<tr>
<td>6: 18.97 x 21.62</td>
<td>600</td>
</tr>
<tr>
<td>7: 23.29 x 30.05</td>
<td>700</td>
</tr>
<tr>
<td>8: 52.37 x 15.28</td>
<td>800</td>
</tr>
<tr>
<td>9: 20.29 x 44.35</td>
<td>900</td>
</tr>
<tr>
<td>10: 35.17 x 28.43</td>
<td>1000</td>
</tr>
</tbody>
</table>

Figure A.12. Illustration of Zoom Function.
<table>
<thead>
<tr>
<th></th>
<th>Alternate 1</th>
<th>Alternate 2</th>
<th>Alternate 3</th>
<th>Alternate 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (4)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>E (3)</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>I (2)</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>O (1)</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>U (0)</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>X (-1)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>32</td>
<td>34</td>
<td>32</td>
<td>33</td>
</tr>
</tbody>
</table>

Figure A.13. Illustration of Step 5 Evaluate Alternative Arrangements.
APPENDIX B

COMPLAN'S SOURCE CODE
COMPLAN is developed on Delphi 3.0 programming. There are a lot of unit in this program but the major unit is main unit. Many units, functions and procedures are gathered up. After combined all unit and complied the whole program, the program is convert to execute program.

This following source code is in main unit which is the most important unit for COMPLAN. Most of functions and procedures are located in this unit including input function, calculation function and others. There are many other units in COMPLAN but less important then it is neglect to illustrate.

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, ExtCtrls, StdCtrls, Menus, Grids, ComCtrls, IniFiles, ImgList;

type array20 = array[1..20] of integer;
  array11_2D = array[1..11, 1..11] of integer;
  array15_2D = array[1..15, 1..15] of integer;

type
  TForm1 = class(TForm)
    ActName1: TLabel;
    ActName2: TLabel;
    ActName3: TLabel;
    ActName4: TLabel;
    ActName5: TLabel;
    ActName6: TLabel;
    ActName7: TLabel;
    ActName8: TLabel;
    ActName9: TLabel;
    ActName10: TLabel;
    ActName11: TLabel;
    ActName12: TLabel;
    ActName13: TLabel;
    ActName14: TLabel;
    ActName15: TLabel;
    ActName16: TLabel;
    ActName17: TLabel;
    ActName18: TLabel;
    ActName19: TLabel;
    ActName20: TLabel;
    AddBtn: TButton;
    Arealm: TEdit;
    Bevel1: TBevel;
    Bevel2: TBevel;
    CtrBtn: TButton;
  end;

{

Sym6: TImage;
Sym7: TImage;
Sym8: TImage;
Sym9: TImage;
Sym10: TImage;
Sym11: TImage;
Sym12: TImage;
Sym13: TImage;
Sym14: TImage;
Sym15: TImage;
Sym16: TImage;
Sym17: TImage;
Sym18: TImage;
Sym19: TImage;
Sym20: TImage;
TabSheet1: TTabSheet;
OpenDialog1: TOpenDialog;
New1: TMenuItem;
Open1: TMenuItem;
Save1: TMenuItem;
SaveAS1: TMenuItem;
N1: TMenuItem;
PrintSetup1: TMenuItem;
Print1: TMenuItem;
N2: TMenuItem;
Exit1: TMenuItem;
Button1: TButton;
lbTree: TLabel;
GroupBox2: TGroupBox;
Manuall: TSpeedButton;
Manualsec: TSpeedButton;
Manual3: TSpeedButton;
Manual4: TSpeedButton;
View1: TMenuItem;
Config1: TMenuItem;
Head5: TLabel;
About1: TMenuItem;
EstabView: TMenuItem;
ChartView: TMenuItem;
DiaView: TMenuItem;
DrawView: TMenuItem;
EvalView: TMenuItem;
ImproveBtn: TButton;
lbScore1: TLabel;
lbScore2: TLabel;
lbScore3: TLabel;
lbScore4: TLabel;
ListBox2: TListBox;
CorShow: TLabel;
Button2: TButton;
SaveDialog1: TSaveDialog;
Button3: TButton;
Button4: TButton;
Button5: TButton;
ImageList2: TImageList;
ImproveCheck: TCheckBox;
N3: TMenuItem;
TCRTabe1: TMenuItem;
ColorCheckBox: TCheckBox;
PrinterSetupDialog1: TPrinterSetupDialog;
procedure AddBtnClick(Sender: TObject);
procedure AreaInKeyPress(Sender: TObject; var Key: Char);
procedure ClrBtnClick(Sender: TObject);
procedure DelBtnClick(Sender: TObject);
procedure FormCreate(Sender: TObject);
procedure FunctionlnChange(Sender: TObject);
procedure LB1Click(Sender: TObject);
procedure LB1DragDrop(Sender, Source: TObject; X, Y: Integer);
procedure LB1DragOver(Sender, Source: TObject; X, Y: Integer; State: TDragState; var Accept: Boolean);
procedure LB1DragOver(Sender, Source: TObject; X, Y: Integer; State: TDragState; var Accept: Boolean);
procedure LB1EndDrag(Sender, Target: TObject; X, Y: Integer);
procedure LB1MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure LB1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
procedure LB1MouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure Notebook1PageChanged(Sender: TObject);
procedure PageControl1Change(Sender: TObject);
procedure Step1Click(Sender: TObject);
procedure ListView1Click(Sender: TObject);
procedure ListView1Deletion(Sender: TObject; Item: TListItem);
procedure ListView1Enter(Sender: TObject);
procedure InsBtnClick(Sender: TObject);
procedure Exit1Click(Sender: TObject);
procedure Open1Click(Sender: TObject);
procedure ImproveBtnClick(Sender: TObject);
procedure Manual1Click(Sender: TObject);
procedure EvalTableDrawCell(Sender: TObject; vCol, vRow: LongInt; Rect: TRect; State: TGridDrawState);
procedure Config1Click(Sender: TObject);
procedure DiagramPic1DrawCell(Sender: TObject; vCol, vRow: Integer; Rect: TRect; State: TGridDrawState);
procedure DiagramPic1MouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure DiagramPic1MouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
procedure DiagramPic1SelectCell(Sender: TObject; Col, Row: Integer; var CanSelect: Boolean);
procedure Step4Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Save1Click(Sender: TObject);
procedure TCRTable1Click(Sender: TObject);
procedure ColorCheckBoxClick(Sender: TObject);
procedure Print1Click(Sender: TObject);
procedure PrintSetup1Click(Sender: TObject);
procedure New1Click(Sender: TObject);
procedure CheckBox2Click(Sender: TObject);
procedure ListView1DblClick(Sender: TObject);
procedure UpdBtnClick(Sender: TObject);
procedure SavePicture1Click(Sender: TObject);
procedure ZoomPicture1Click(Sender: TObject);
procedure EstabViewClick(Sender: TObject);
procedure ChartViewClick(Sender: TObject);
procedure DiaViewClick(Sender: TObject);
procedure DrawViewClick(Sender: TObject);
procedure EvalViewClick(Sender: TObject);
procedure About1Click(Sender: TObject);

private
{ Private declarations }

public
{ Public declarations }

procedure DrawRoom(Sender: TObject);
procedure ClearRoom(Sender: TObject);
procedure DrawTree(Sender: TObject);
procedure DrawRect(Sender: TObject; xl,yl,x2,y2: Integer);
procedure SplitTable(Sender: TObject;
  Var arr1,arr2 : array20;
  Var countl,count2: integer;
  Var xl1,yl1,x12,y12: integer;
  Var x21,y21,x22,y22 : integer);
procedure FillColor(Sender: TObject; max: integer);
procedure ShowDetail(Sender: TObject);
procedure ShowLayout(Sender: TObject);
procedure Improve(Sender: TObject; no: integer);
procedure DrawBoard(Sender: TObject);
procedure GenTree2(Sender: TObject);
procedure ImportBoard(Sender: TObject);
procedure CheckArrangement(col: integer);
procedure ClearAll;
procedure PrintRolChart(Sender: TObject);
procedure PrintLayout(no: integer);
end;

Const
ManName: Array[0..4] of String[6] = ('No.bmp','UL.bmp','UR.bmp','LL.bmp','LR.bmp');

FontColor: Array[-1..4] of TColor = (clBlack,clWhite,clBlue,clGreen,clYellow,clRed);

StChar: array[-1..4] of Char = ('X','U','O','I','E','A');
UserWeight: array[-1..4] of integer = (-1,0,1,2,3,4);
StdWeight : array[-1..4] of integer = (-1,0,1,2,3,4);
MaxRoom = 20;
CentroidX : array[1..20] of Extended = (0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5,9.5,10.5,11.5,12.5,13.5,14.5,15.5,16.5,17.5,18.5,19.5);
CentroidY : array[1..20] of Extended = (0.5,1.5,2.5,3.5,4.5,5.5,6.5,7.5,8.5,9.5,10.5,11.5,12.5,13.5,14.5,15.5,16.5,17.5,18.5,19.5);

type TTable = array [1..MaxRoom,1..MaxRoom] of Integer;
BoardArray = array[1..11,1..11] of integer;

type CoordRec = record
    no : integer;
    x1,x2,y1,y2 : integer;
end;

var Form1: TForm1;
    board : BoardArray;
    ActNum, LBCount, LBACount, LBECount, LBICount, LBOCount, LBUCount, LBXCount : Integer;
    BoldFlag: Boolean= False;
    LBTable,
        RSTable: TTable;
    img: array[1..20] of integer;
    LBCaption:
        String = 'A';
    LBTag: Integer= 4;
    ReasonFlag: Boolean =False;
    RS
        Tag: integer = 1;
    SumTCR: Integer;
    MaxTable : Integer;
    ReasonSt:
        array[0 .. 20]
            of String;
    RoomName:
        array[0 .. MaxRoom]
            of String;
    RoomArea,Area,Func:
        array[0 .. MaxRoom]
            of Integer;
    Flag : array[1..MaxRoom]
        of Boolean;
    Sort : array[1..MaxRoom]
        of Integer;
    Cal : array[1..11,1..11]
        of Extended;
    Sortx,Sorty,RoomX,RoomY : array[1..20]
        of integer;
    StdTCR,UserTCR:
        array[1..MaxRoom]
            of Integer;
    ActiveDiagram : Integer;
    SelCol,SelRow,OldCol,OldRow
        : Integer;
    MoveText : Integer;
    Adj,Dist,Calc : array[1..20,1..20]
        of integer;
    NEWTCR : array[1..20]
        of Integer;
    Mark : Extended;
    ImproveMethod : Integer = 0;
    GenMethod : Integer = 0;
    NewData : Boolean = True;
    PrintFlag : Boolean = False;
    bCoord : array[1..20]
        of CoordRec;
implementation

uses relNo, config, showPic, Unit6, Unit7, print, Unit8, About;   
{$R *.DFM} 

function CalcDistance(i,j : Integer): Extended;
var d,dx,dy : extended;
begin
    dx := (RoomX[i]-RoomX[j])*(RoomX[i]-RoomX[j]);
    dy := (RoomY[i]-RoomY[j])*(RoomY[i]-RoomY[j]);
    d := Sqrt(dx+dy);
    Result := d;
end;
function CalcLB(n : Integer): Integer;
begin
    CalcLB := (n * (n-1)) div 2;
end;

function Order(n : Integer): Integer;
begin
    if n = 0 then order := 0 else order := n + order(n-1);
end;

procedure ArrayDel(i: Integer; var Arr: TTable);
var Col, Row: Integer;
begin
    for Col := i to 19 do
        for Row := 1 to 20 do
            Arr[Col,Row] := Arr[Col+1,Row]; {Shift Left}
    for Row := i to 19 do
        for Col := 1 to 20 do
            Arr[Col,Row] := Arr[Col,Row+1]; {Shift up}
    for Col := 1 to 20 do
        Arr[Col,20] := 0; {Last row clear}
    for Row := 1 to 20 do
        Arr[20,Row] := 0; {Last col clear}
    for Col := 1 to 20 do
        for Row := 1 to 20 do
            if Col = Row then Arr[Col,Row] := 0;
end;

procedure ArrayIns(i: Integer; var Arr: TTable);
var Col, Row: Integer;
begin
    for Col := 1 to 20 do
        Arr[Col,20] := 0; {Clear last row}
    for Row := 1 to 20 do
        Arr[20,Row] := 0; {Clear last Col}
    for Col := 20 downto i+1 do
        for Row := 1 to 20 do
            Arr[Col,Row] := Arr[Col-1,Row]; {Shift right}
    for Row := 20 downto i+1 do
        for Col := 1 to 20 do
            Arr[Col,Row] := Arr[Col,Row-1]; {Shift down}
    for Col := 1 to 20 do
        Arr[Col,Row] := Arr[Row,Col];
    for Col := 1 to 20 do
        Arr[Col,i] := 0; {Inserted row clear}
    for Row := 1 to 20 do
        Arr[i,Row] := 0; {Inserted col clear}
    for Col := 1 to 20 do
        for Row := 1 to 20 do
            if Col = Row then Arr[Col,Row] := 0;
end;

procedure CalcTCR;
var i, j, Value: Integer;
SumA, SumE, SumI, SumO, SumU, SumX: Integer;
begin
    LBACount := 0;
    LBECount := 0;
    LBICount := 0;
    LBOCount := 0;
    LBUCount := 0;
    LBXCount := 0;
    for j := 1 to ActNum do
        begin
            SumA := 0; SumE := 0; SumI := 0;
            SumO := 0; SumU := 0; SumX := 0;
            for i := 1 to ActNum do
                \...
begin
if (i<>j) then
begin
  Value := LBTable[i,j];
  case Value of
    4 : Inc(SumA);
    3 : Inc(SumE);
    2 : Inc(SumI);
    1 : Inc(SumO);
    0 : Inc(SumU);
    -1 : Inc(SumX);
  end;
end;
end;

               SumU*UserWeight[0] + SumX*UserWeight[-1];
            SumI*StdWeight[2] + SumO*StdWeight[1] +
            SumU*StdWeight[0] + SumX*StdWeight[-1];
LBACount := LBACount + SumA; LBECount := LBECount + SumE;
LBICount := LBICount + SumI; LBOCount := LBOCount + SumO;
LBUCount := LBUCount + SumU; LBXCount := LBXCount + SumX;
end;

LBACount := LBACount div 2; LBECount := LBECount div 2;
LBICount := LBICount div 2; LBOCount := LBOCount div 2;
LBUCount := LBUCount div 2; LBXCount := LBXCount div 2;
SumTCR := 0;
for i := 1 to ActNum do
  SumTCR := SumTCR + StdTCR[i];
SumTCR := SumTCR div 2;

Form2.StringGrid1.Cells[2,1] := IntToStr(LBACount);
Form2.StringGrid1.Cells[2,2] := IntToStr(LBECount);
Form2.StringGrid1.Cells[2,3] := IntToStr(LBICount);
Form2.StringGrid1.Cells[2,4] := IntToStr(LBOCount);
Form2.StringGrid1.Cells[2,5] := IntToStr(LBUCount);
end;

procedure NumtoCor(n: integer; var col,row :integer);
var i,x: integer;
begin
  x:=0;
  for i:= 1 to 20 do
    if order(i) < n then x := i;
  col:= x + 2;
  row:= n - order(x);
end;

procedure TForm1.AddBtnClick(Sender: TObject);
var V,Code : integer;
begin
  Val(AreaIn.Text, V,Code);
  if (V=0) then
    begin
      ShowMessage('Area is zero!');
      exit;
    end;
end;
end;
Itemcount := ListView1.Items.Count;
If (ItemCount < 20) and (Code = 0) then
begin
If (NameIn.Text <> "") and (AreaIn.Text <> ") then
begin
With ListView1.Items.add do
begin
Caption := NameIn.Text;
Subitems.add(AreaIn.Text);
ImageIndex := FunctionIn.ItemIndex; {Change ImPreview when add}
end;
Itemcount := ListView1.Items.Count;
if (ItemCount > 1) then begin
Step2.Enabled := True;
ChartView.Enabled := True;
end;
end;
end;
ClrBtn.Enabled := True;
end;
NameIn.Clear;
AreaIn.Clear;
// ActFunction.ItemIndex := 0; {Reset ActFunction index to 0}
// ActFunctionChange(self); {Reset ImPreview to 0}
If ListView1.Items.Count = 20 then AddBtn.Enabled := False;
NameIn.SetFocus;
NewData := True;
end;
end;

procedure TForm1.AreaInKeyPress(Sender: TObject; var Key: Char);
begin
If not (Key in ['0'..'9',#8]) then Key := #0;
end;

procedure TForm1.ClrBtnClick(Sender: TObject);
begin
If MessageDlg('All data will be deleted', mtConfirmation, [mbYes,mbNo],0) = mrYes then
begin
ListView1.Items.Clear;
NameIn.SetFocus;
AddBtn.Enabled := True;
ClrBtn.Enabled := False;
ClearAll;
Step2.Enabled := False;
Step3.Enabled := False;
Step4.Enabled := False;
Step5.Enabled := False;
ChartView.Enabled := False;
DiaView.Enabled := False;
DrawView.Enabled := False;
EvalView.Enabled := False;
TCRTable1.Enabled := False;
end;
end;

procedure TForm1.DelBtnClick(Sender: TObject);
var Index,ItemCount : integer;
begin
Index := ListView1.ItemFocused.Index + 1;
ListView1.ItemFocused.Delete;
procedure TForm1.FunctionlnChange(Sender: TObject);
var i: integer;
aBitmap: TBitmap;
begin
  i := Functionln.ItemIndex;
aBitmap := TBitmap.Create;
  ImageList2.GetBitmap(i,aBitmap);
  ImPreview.picture.assign(aBitmap);
  aBitmap.Free;
end;

procedure TForm1.FormCreate(Sender: TObject);
var Col,Row: integer;
abitmap: Tbitmap;
begin
  Functionln.ItemIndex := 0;
  aBitmap := TBitmap.Create;
  ImageList2.GetBitmap(Functionln.ItemIndex,aBitmap);
  ImPreview.picture.assign(aBitmap);
  aBitmap.Free;
  for Col := 1 to 20 do
  begin
    for Row := 1 to 20 do begin
      LBTable[Col,Row] := 0;
      RSTable[Col,Row] := 0;
    end;
  end;
  EvalTable.Cells[0,1] := 'A (4)';
  EvalTable.Cells[0,2] := 'E (3)';
  EvalTable.Cells[0,3] := 'I (2)';
  EvalTable.Cells[0,4] := 'O (1)';
  EvalTable.Cells[0,5] := 'U (0)';
  EvalTable.Cells[0,6] := 'X (-1)';
  EvalTable.Cells[0,7] := 'Total';
  EvalTable.Cells[1,0] := 'Alternate 1';
  EvalTable.Cells[2,0] := 'Alternate 2';
  EvalTable.Cells[3,0] := 'Alternate 3';
  EvalTable.Cells[4,0] := 'Alternate 4';
  NewData := True;
end;

procedure TForm1.FormCreate(Sender: TObject);
var Col,Row: integer;
abitmap: Tbitmap;
begin
  Functionln.ItemIndex := 0;
  aBitmap := TBitmap.Create;
  ImageList2.GetBitmap(Functionln.ItemIndex,aBitmap);
  ImPreview.picture.assign(aBitmap);
  aBitmap.Free;
end;
procedure TForm1.LB1Click(Sender: TObject);
Var Value : Integer;
   Col,Row : Integer;
begin
   // onclick //
   Value := TLabel(Sender).Tag;
   NumToCor(Value,Col,Row);
   Value := LBTable[Col,Row];
   Value := Value+1;
   if Value > 4 then Value := -1;
   LBTable[Col,Row] := Value;
   LBTable[Row,Col] := Value;
   TLabel(Sender).Caption := StChar[Value];
   if ColorCheckBox.Checked then
      TLabel(Sender).Font.Color := FontColor[LBTable[Col,Row]]
   else
      TLabel(Sender).Font.Color := ClWhite;
   CalcTCR;
end;

procedure TForm1.LB1DblClick(Sender: TObject);
begin
   // onDB
end;

procedure TForm1.LB1DragDrop(Sender, Source: TObject; X, Y: Integer);
begin
   // onDrag
end;

procedure TForm1.LB1DragOver(Sender, Source: TObject; X, Y: Integer;
   State: TDragState; var Accept: Boolean);
begin
   // ondrag
end;

procedure TForm1.LB1EndDrag(Sender, Target: TObject; X, Y: Integer);
begin
   // ondragCM
end;

procedure TForm1.LB1MouseDown(Sender: TObject; Button: TMouseButton;
   Shift: TShiftState; X, Y: Integer);
begin
   // onmouse
end;

procedure TForm1.LB1MouseMove(Sender: TObject; Shift: TShiftState; X,
   Y: Integer);
begin
   If (Sender is TLabel) then
      begin
         CorShow.Caption := 'Dept.' + TLabel(Sender).Hint;
         RelShow.Caption := 'Rel.' + TLabel(Sender).Caption;
      end
   else
      begin
         CorShow.Caption := ";
         RelShow.Caption := ";
      end;
end;


procedure TForm1.LBMMeUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
// var Col, Row : Integer;
begin
(* NumToCor(TLabel(Sender).Tag, Col, Row);
   if ReasonFlag then
      begin
         TLabel(Sender).Caption := IntToStr(RSTag);
         RSTable[Col, Row] := RSTag; // update RSTable
         RSTable[Row, Col] := RSTag; // update RSTable
      end else begin
         TLabel(Sender).Font.Color := FontColor[LBTag];
         TLabel(Sender).Caption := LBCaption;
         LBTable[Col, Row] := LBTag; // Update LBTable
         LBTable[Row, Col] := LBTag; // Update LBTable
      end;
      CalcTCR; *)
end;

procedure TForm1.LBMStartDrag(Sender: TObject);
var DragObject: TDragObject;
// onmouse
end;

procedure TForm1.LBPickAClick(Sender: TObject);
var FindLB : TComponent;
i, Col, Row : integer;
Down : Boolean;
begin
   // LBCaption := TSpeedButton(Sender).Caption;
   Down := TSpeedButton(Sender).Down;
   LBTag := TSpeedButton(Sender).Tag;
   For i := 1 to LBCount do
      begin
         NumToCor(i, Col, Row);
         if (LBTable[Col, Row] = LBTag) then
            begin
               FindLB := FindComponent('LB' + IntToStr(i));
               if Down then
                  TLabel(FindLB).Font.Color := FontColor[LBTag];
                  TLabel(FindLB).Font.Style := TLabel(FindLB).Font.Style + [fsBold]
               else
                  TLabel(FindLB).Font.Color := ClWhite;
                  TLabel(FindLB).Font.Style := TLabel(FindLB).Font.Style - [fsBold]
            end;
      end;
end;

procedure TForm1.LBPickAMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
// var i : integer;
// FindLB : TComponent;
begin
   { TSpeedButton(Sender).Font.Color := FontColor[TSpeedButton(Sender).Tag];
      For i := 1 to 190 do
      begin
         FindLB := FindComponent('LB' + IntToStr(i));
         if TLabel(FindLB).Caption = TSpeedButton(Sender).Caption

then begin
  TLabel(FindLB).Font.Style := TLabel(FindLB).Font.Style + [fsBold];
  TLabel(FindLB).Font.Color := TSpeedButton(Sender).Font.Color;
end else begin
  TLabel(FindLB).Font.Style := TLabel(FindLB).Font.Style - [fsBold];
  TLabel(FindLB).Font.Color := clWhite;
end;
// want mousedown again font.style will be -[fsBold]
end;

procedure TForm1.Notebook1PageChanged(Sender: TObject);
begin
  if (NoteBook1.PageIndex = 1) then begin
    if checkbox2.Checked then begin
      Form2.Show;
    end;
    end else if checkbox2.checked then begin
      Form2.Close;
    end;
  end;

procedure TForm1.Notebook1PageChanged(Sender: TObject);
begin
  if ((Sender as TPageControl).ActivePage = TabSheet1)
  then ReasonFlag := False
  else ReasonFlag := True;
if ReasonFlag = True then begin
  Form1.SetFocus
end else begin
  Form2.Show;
  Form1.SetFocus
end;

for i := 1 to 190 do begin
  NumToCor(i, Col, Row);
  FindLB := FindComponent('LB' + IntToStr(i));
  if ReasonFlag then begin
    TLabel(FindLB).Caption := IntToStr(RSTable[Col, Row]);
  end else begin
    TLabel(FindLB).Caption := StChar[LBTable[Col, Row]];
    if ColorCheckBox.Checked then
      TLabel(FindLB).Font.Color := FontColor[LBTable[Col, Row]]
    else
      TLabel(FindLB).Font.Color := ClWhite;
  end;
end;

procedure TForm1.Step1Click(Sender: TObject);
begin
  NoteBook1.PageIndex := TSpeedButton(Sender).Tag;
end;
procedure TForm1.Step2Click(Sender: TObject);
var col, row, i : integer;
FindLB, FindIm : TComponent;
aBitmap : TBitmap;
begin
  // if NewData then
  // begin
  ActNum := ListView1.Items.Count;
  MaxTable := ActNum;
  LBCount := CalcLB(ActNum);
  For i := 0 to ListView1.Items.Count-1 do
  begin
    img[i+1] := ListView1.Items[i].ImageIndex;
    Func[i] := img[i+1];
    RoomName[i] := ListView1.Items[i].Caption;
    RoomArea[i] := StrTolnt(ListView1.Items[i].SubItems[0])*10;
  end;

  For i := 1 to 20 do {Clear all}
  begin
    FindIm := FindComponent('Sym'+ InttoStr(i)) as TImage;
    TImage(FindIm).picture := nil;
    TImage(FindIm).Visible := False;
    FindLB := FindComponent('ActName'+ InttoStr(i));
    TLabel(FindLB).Caption := "";
    TLabel(FindLB).Visible := False;
  end;

  For i := 1 to ActNum do {ActNum 1 to 20}
  begin
    aBitmap := TBitmap.Create;
    FindIm := FindComponent('Sym'+ InttoStr(i)) as TImage;
    ImageList1.GetBitmap(img[i], aBitmap);
    TImage(FindIm).picture.assign(aBitmap);
    TImage(FindIm).Visible := True;
    FindLB := FindComponent('ActName'+ InttoStr(i));
    TLabel(FindLB).Caption := IntToStr(i) + ': ' + ListView1.Items[i-1].Caption; {Show ActName in RelChart}
    TLabel(FindLB).Visible := True;
    aBitmap.Free;
  end;

  For i := 1 to 190 do {Clear all LB}
  begin
    FindLB := FindComponent('LB'+ InttoStr(i));
    TLabel(FindLB).Caption := 'U';
    TLabel(FindLB).Visible := False;
  end;

  For i := 1 to LBCount do {LBCount 1 to 190}
  begin
    FindLB := FindComponent('LB'+ InttoStr(i));
    TLabel(FindLB).Visible := True;
    NumToCor(i, Col, Row);
    TLabel(FindLB).Caption := StChar[LBTable[Col,Row]]; {Load caption from LBTable}
    if ColorCheckBox.Checked then
      TLabel(FindLB).Font.Color := FontColor[LBTable[Col,Row]]
    else
      TLabel(FindLB).Font.Color := CLWhite;
  end;
end;
CalcTCR;
Step3.Enabled := True;
DiaView.Enabled := True;
TCRTable1.Enabled := True;
NewData := False;

// end;
NoteBook1.PageIndex := TSpeedButton(Sender).Tag;
end;

procedure TForm1.ListView1Click(Sender: TObject);
begin
If ListView1.ItemFocused <> Nil then
begin
InsBtn.Enabled := True;
DelBtn.Enabled := True;
end;
end;

procedure TForm1.ListView1Deletion(Sender: TObject; Item: TListItem);
begin
InsBtn.Enabled := False;
DelBtn.Enabled := False;
ListView1.ItemFocused := Nil;
end;

procedure TForm1.ListView1Enter(Sender: TObject);
begin
ListView1.ItemFocused := Nil;
end;

procedure TForm1.InsBtnClick(Sender: TObject);
var V,Code,index : integer ;
begin
Index := ListView1.ItemFocused.Index+1; {Convert to (1 .. 20) array}
Val(AreaIn.Text,V,Code);
If(ListView1.Items.Count < 20) and (Code = 0) then
begin
If(Nameln.Text <>")and (AreaIn.Text <>")then
begin
With ListView1.Items.Insert(Index-1) do {Convert to (0 .. 19) array}
begin
Caption:= Nameln.Text;
Subitems.add(AreaIn.Text);
ImageIndex := FunctionIn.ItemIndex; {Change ImPreview when add}
end;
ArrayIns(Index,LBTable);
ArrayIns(Index,RSTable);
end;
ClrBtn.Enabled :=True;
end;
Nameln.Clear;
AreaIn.Clear;
// ActFunction.ItemIndex := 0; {Reset ActFunction index to 0}
// ActFunctionChange(self); {Reset ImPreview to 0}
If(ListView1.Items.Count = 20 then AddBtn.Enabled := False;
Nameln.SetFocus;
end;

procedure TForm1.ExitlClick(Sender: TObject);
begin

procedure LoadAreaFromFile(FileName: String; No: Integer);
Var myini : TlniFile;
i : integer;
Ident : String;
begin
  myini := TiniFile.Create(FileName);
  for i := 0 to No-1 do
    begin
      Ident := 'Area'+IntToStr(i);
      RoomArea[i] := myini.ReadInt('Area',Ident,0)*10;
    end;
  myini.free;
end;

procedure LoadFunctionFromFile(FileName: String; No: Integer);
Var myini : TlniFile;
i : integer;
Ident : String;
begin
  myini := TiniFile.Create(FileName);
  for i := 0 to No-1 do
    begin
      Ident := 'F'+IntToStr(i);
      Func[i] := myini.ReadInt('Function',Ident,0);
    end;
  myini.free;
end;

procedure LoadNameFromFile(FileName : String; No : Integer);
Var myini : TlniFile;
i : integer;
Ident : String;
begin
  myini := TiniFile.Create(FileName);
  for i := 0 to No-1 do
    begin
      Ident := 'Name'+IntToStr(i);
      RoomName[i] := myini.ReadString('Name',Ident, 'none');
    end;
  myini.free;
end;

procedure LoadReasonFromFile(FileName: String; MaxTable: Integer);
Var myini : TlniFile;
i : integer;
Ident : String;
begin
  myini := TiniFile.Create(FileName);
  for i := 0 to MaxTable-1 do
    begin
      Ident := 'Reason'+IntToStr(i);
      ReasonSt[i] := myini.ReadString('Reason',Ident, 'none');
    end;
  myini.free;
end;

function CharToInt(Ch : String) : Integer;

Var Ret : Integer;
begin
    Ch := UpperCase(Ch);
    if Ch = 'U' then Ret := 0
    else if Ch = 'O' then Ret := 1
    else if Ch = 'E' then Ret := 3
    else if Ch = 'A' then Ret := 4
    else if Ch = '@' then Ret := 5
    else Ret := -1;
    Result := Ret;
end;

procedure LoadTableFromFile(FName : String);
Var Txt : TextFile;
Line,Str : String;
department,Code,i,j : Integer;
begin
    AssignFile(Txt,FName);
    Reset(Txt);
    ReadLn(Txt,Line);
    Line:= Trim(Line);
    if(Pos('dept=',Line)>0) then
        begin
            Val(Copy(Line,6,3),department,Code);
            if (department = 0) or (department > 20) then
                begin
                    ShowMessage('Error: Counter Error or Over 20 [Dept=xx]');
                    CloseFile(txt);
                    Exit;
                end;
            end else
                begin
                    ShowMessage('Error: Can“t Read [Dept=xx]');
                    CloseFile(txt);
                    Exit;
                end;
    Form1.ClearAll;
    for j := 1 to department do
        begin
            ReadLn( txt,Line);
            Line:= Trim(Line);
            for i := 1 to department do
                begin
                    Str := Line[(i*2)-1];
                    LBTable[i,j] := CharToInt(Str);
                end;
        end;
    ReadLn(txt);
    for j := 1 to department do
        begin
            ReadLn( txt,Line);
            Line:= Trim(Line);
            for i := 1 to department do
                begin
                    if i <> j then
                        begin
                            Str := Line[(i*2)-1];
                            RSTable[i,j] := StrToInt(Str);
                        end;
                end;
        end;
end;
CloseFile(txt);

MaxTable := dept;
LoadAreaFromFile(FName,MaxTable);
LoadNameFromFile(FName,MaxTable);
LoadFunctionFromFile(FName,MaxTable);
Form1.ListView1.Items.Clear;

for i := 0 to MaxTable-1 do
begin
  With Form1.ListView1.Items.add do
  begin
    Caption := RoomName[i];
    Subitems.add(IntToStr(RoomArea[i] div 10));
    ImageIndex := Func[i]; {Change ImPreview when add}
  end;
end;
end;

procedure TForm1.Open1Click(Sender: TObject);
var filename : String;
begin
  if opendialog1.execute then
  begin
    filename := opendialog1.filename;
    LoadTableFromFile(FileName);
    Step1.Enabled := True;
    Step2.Enabled := True;
    Step3.Enabled := False;
    Step4.Enabled := False;
    Step5.Enabled := False;
    EstabView.Enabled := True;
    ChartView.Enabled := True;
    DiaView.Enabled := False;
    DrawView.Enabled := False;
    EvalView.Enabled := False;
    NewData := True;
    NoteBook1.PageIndex := 0;
  end;
end;

procedure AroundArea(x,y: integer);
begin
  if(board[x-1,y]=0) then board[x-1,y] := -1;
  if(board[x+1,y]=0) then board[x+1,y] := -1;
  if(board[x,y-1]=0) then board[x,y-1] := -1;
  if(board[x,y+1]=0) then board[x,y+1] := -1;
end;

procedure Walk(i: integer;var x,y: integer);
begin
  board[x,y] := Sort[i];
  SortX[i] := x;
  Sorty[i] := y;
  RoomX[Sort[i]] := x;
  RoomY[Sort[i]] := y;
  AroundArea(x,y);
Function CalcCenTroidX(X,No : Integer): Extended;
Var i : Integer;
    Sum : Extended;
begin
    Sum := 0.0;
    for i := 1 to No-1 do
    begin
        Sum := Sum+CentroidX[SortX[i]];
    end;
    SUM := Sum+CentroidX[X];
    Sum := Sum/No;
    Result := Sum;
end;

Function CalcCenTroidY(Y,No : Integer): Extended;
Var i : Integer;
    Sum : Extended;
begin
    Sum := 0.0;
    for i := 1 to No-1 do
    begin
        Sum := Sum+CentroidY[SortY[i]];
    end;
    Sum:= Sum+CentroidY[Y] ;
    Sum := Sum/No;
    Result := Sum;
end;

procedure CalWeight(no: integer;var x,y: integer);
Var i,j,k,m: integer;
    Dist, W, dTCR, MinDist, MindTCR, Centroid, MinCentroid : extended;
    Value, MinX, MinY : integer;
    BXCentroid, BYCentroid, Wei, adTCR : array[1..11, 1..11] of Extended;
begin
    MinDist := 999999.0; // initial MinDist to a large number
    for j := 1 to 11 do
    begin
        for i := 1 to 11 do
        begin
            Wei[i,j] := 999999.0;
            adTCR[i,j] := 999999.0;
            if (Board[i,j] = -1) then // if area border
            begin
                W := 0.0; dTCR := 0.0;
                for k := 1 to No-1 do // for sort order
                begin
                    m := k;
                    Dist := sqrt((i-SortX[m])*(i-SortX[m])+(j-SortY[m])*(j-SortY[m]));
                    Value := LBTable[Sort[m],Sort[no]];
                    W := W + (UserWeight[Value]*Dist);
                    dTCR := dTCR + (UserWeight[Value] * Dist);
                end;
                Wei[i,j] := W;
                adTCR[i,j] := dTCR;
                if (W < MinDist) then // find minimum distance area
                begin
                    MinDist := W;
                    MinX := i;
                    MinY := j;
                end;
            end;
        end;
    end;
end;
MindTCR := dTCR;
end else
if (W = MinDist) then // when MinDist ties
begin
if (dTCR < MindTCR) then // find lowest sum dTCR
begin
MinX := i;
MinY := j;
MindTCR := dTCR;
end;
end;
end; // if board[i,j] = -1
end; // end for i
end; // end for j

MinCentroid := 999999.0;
for j := 1 to 11 do
begin
for i := 1 to 11 do
begin
if (Board[i,j] = -1) and (wei[i,j] <= MinDist) and (adTCR[i,j] <= MindTCR) then
begin
BXCentroid[i,j] := CalcCentroidX(i,No);
BYCentroid[i,j] := CalcCentroidY(j,No);
Centroid := sqrt( (CentroidX[i]-BXCentroid[i,j])*(CentroidX[i]-BXCentroid[i,j])+
                  (CentroidY[j]-BYCentroid[i,j])*(CentroidY[j]-BYCentroid[i,j]) )
if (Centroid < MinCentroid) then
begin
MinCentroid := Centroid;
MinX := i;
MinY := j;
end;
end; // end if(board[i,j] = -1)
end; // end for i
end; // end for j

x := MinX;
y := MinY;
end;

procedure TForm1.DrawRoom(Sender: TObject);
var i, j: integer;
begin
for j := 1 to 11 do
begin
for i := 1 to 11 do
begin
if (Board[i,j] > 0) then
  TStringGrid(Sender).Cells[i-1,j-1] := Format("%d",[Board[i,j]])
else begin
  TStringGrid(Sender).Cells[i-1,j-1] := ";
  board[i,j] := 0;
end;
end;
end;
end;

procedure TForm1.ClearRoom(Sender: TObject);
var i, j: integer;
begin

end;
begin
for i := 1 to 11 do 
begin
RoomX[i] := 0;
RoomY[i] := 0;
for j := 1 to 11 do 
begin
board[i][j] := 0;
end;
end;
DrawRoom(Sender);
end;

procedure Gen Tree;
Var i,j,max,index,count : integer;
St : String;
EndLoop : Boolean;
Begin
Form1.ClearRoom(Form1.DiagramPic1);
St := "; EndLoop := False;
// oldpos := 0;
Count := 0;
for i := 1 to 20 do Flag[i] := False;
Index := 1;
Max := UserTCR[Index];
for i := 1 to 20 do Sort[i] := -10000;
for i := 2 to MaxTable do 
begin
if UserTCR[i] > Max then 
begin
index := i;
Max := UserTCR[i];
end;
end;
Flag[Index] := True;
St := IntToStr(Index);
Inc(Count);
Sort[Count] := Index;
Repeat
Max := -30000;
for j := 1 to MaxTable do // scan each room area (column)
begin
if Flag[j] then // if have selected
begin
for i := 1 to MaxTable do // scan member (row)
begin
if not Flag[i] and (i <> j) then 
begin
if (LBTable[j][i] > Max) then 
begin
Max := LBTable[j][i];
Index := i;
end;
end;
end;
end;
end;
end;
Flag[Index] := True;
Inc(Count);
Sort[Count] := Index;
St := St + '"->' + IntToStr(Index);
for i := 1 to MaxTable do
begin
  if not Flag[i] then begin
    EndLoop := False;
    Break;
  end else
    EndLoop := True;
end;
Until Endloop;

Form1.lbTree.Caption := 'Result = ' + St;
end;

procedure TForm1.DrawTree(Sender : TObject);
var x,y,i : integer;
begin
  ClearRoom(Sender as TStringGrid);
  x := 6; y := 6;
  Walk(1,x,y);
  for i := 2 to MaxTable do
    begin
      CalWeight(i,x,y);
      Walk(i,x,y);
    end;
  DrawRoom(Sender as TStringGrid);
end;

procedure TForm1.GenTree2(Sender : TObject);
Var List1 : array[1..20] of Integer;
    GenBoard : array[1..11, 1..11] of Integer;
    Value,Col,Row,i,j : Integer;
begin
  for i := 1 to 11 do
    for j := 1 to 11 do
      GenBoard[i,j] := 0; // clear GenBoard
  for i := 1 to MaxTable do
    begin
      Repeat
        Col := Random(7)+2;
        Row := Random(7)+2;
        Value := GenBoard[Col,Row];
        Until (Value = 0); // until empty position
      GenBoard[Col,Row] := i;
    end;
  Value := 0;
  for j := 1 to 11 do
    begin
      for i := 1 to 11 do
        begin
          if GenBoard[i,j] <> 0 then
            begin
              Inc(Value);
              List1[Value] := GenBoard[i,j];
            end;
        end;
    end;
end;
if (GenMethod = 0) then
begin
  for i := 1 to MaxTable do
    Sort[i] := List1[i];
  ClearRoom(Sender);
  DrawTree(Sender);
end
else begin
  for j := 1 to 11 do
    begin
      for i := 1 to 11 do
        begin
          Board[i,j] := 0;
          Value := GenBoard[i,j];
          if (Value > 0) then
            begin
              Board[i,j] := Value;
              RoomX[Value] := i;
              RoomY[Value] := j;
            end;
        end;
    end;
  DrawBoard(Sender);
end;

procedure TForm1.Button1Click(Sender: TObject);
Var StopTimer, StartTimer: TTime;
beg
  Step4.Enabled := True;
  Step5.Enabled := True;
  DrawView.Enabled := True;
  EvalView.Enabled := True;
  Randomize();
  GenTree1;
  DrawTree(DiagramPic1);
  // ShowMessage('OK');
  StartTimer := GetTickCount;
  Improve(DiagramPic1,1);
  StopTimer := GetTickCount;
  lbTime1.Caption := Format('%0.2f Sec',[(StopTimer - StartTimer)/1000]);
  ShowLayout(LayoutPic1);
  CheckArrangement(1);

  GenTree2(DiagramPic2);
  StartTimer := GetTickCount;
  Improve(DiagramPic2,2);
  StopTimer := GetTickCount;
  lbTime2.Caption := Format('%0.2f Sec',[(StopTimer - StartTimer)/1000]);
  ShowLayout(LayoutPic2);
  CheckArrangement(2);

  GenTree2(DiagramPic3);
  StartTimer := GetTickCount;
  Improve(DiagramPic3,3);
  StopTimer := GetTickCount;
  lbTime3.Caption := Format('%0.2f Sec',[(StopTimer - StartTimer)/1000]);
  ShowLayout(LayoutPic3);
CheckArrangement(3);
GenTree2(DiaGramPic4);
StartTimer := GetTickCount;
Improve(DiagramPic4,4);
StopTimer := GetTickCount;
lbTime4.Caption := Format('%0.2f Sec',[StopTimer-StartTimer)/1000]);
ShowLayout(LayoutPic4);
CheckArrangement(4);

Const ColorTable : array[1..20] of TColor =
(clBlack,clMaroon,clOlive,clNavy,clPurple,clTeal,clFuchsia,clGray,clRed,clBlue, clBlack,clMaroon,clOlive,clNavy,clPurple,clTeal,clFuchsia,clGray,clRed,clBlue);

Var
  SumRow,SumCol : array[1..11] of integer;
  arr1,arr2,arr3,arr4,arr5,arr6,arr7,arr8,arr9 : array[20];
  SumArea,Count,x1,y1,x2,y2 : array[1..20] of Integer;
  Table2,Tb2 : array[1..11,1..11] of integer;

  Table : array[1..11,1..11] of Integer;
  Scale : Real = 2.5;

procedure TForm1.DrawRect(Sender : TObject; x1,y1,x2,y2 : Integer);
begin
  with Timage(Sender) do
  begin
    Canvas.moveto(Round(x1*Scale),Round(y1*Scale));
    Canvas.lineto(Round(x2*Scale),Round(y1*Scale));
    Canvas.lineto(Round(x2*Scale),Round(y2*Scale));
    Canvas.lineto(Round(x1*Scale),Round(y2*Scale));
    Canvas.lineto(Round(x1*Scale),Round(y1*Scale));
  end;
end;

function ScanTable(Table: array[1..20] of Integer; Var x,y: integer): Char;
var
  i,j: integer;
  SumC,SumR,SumArea,MidArea,Row,Col,Temp : Integer;
begin
  for i := 1 to 11 do begin
    SumRow[i] := 0;
    SumCol[i] := 0;
  end;
  for j := 1 to 11 do
  begin
    for i := 1 to 11 do
      begin
        SumRow[j] := SumRow[j] + Area[Table[i,j]];
      end;
  end;
  // --- start calc sum each row
  for j := 1 to 11 do
  begin
    for i := 1 to 11 do
      begin
        SumRow[j] := SumRow[j] + Area[Table[i,j]];
      end;
  end;
  // start calc sum each column
  for j := 1 to 11 do
  begin
    for i := 1 to 11 do
begin
  SumCol[i] := SumCol[i] + Area[Table[i,j]];
end;
end;

// start calc sum all area
SumArea := 0;
for i := 1 to 11 do
  SumArea := SumArea + SumCol[i];
MidArea := SumArea div 2;

// find middle sum row
Temp := 0; Row := 0; SumR := 0;
for i := 1 to 11 do
begin
  Temp := SumR + SumRow[i];
  Row := i;
  if (Temp >= MidArea) then break
  else SumR := Temp;
end;
if (MidArea - SumR) < (Temp - MidArea) then
  Row := Row - 1
else
  SumR := Temp;

// start find middle sum column
Temp := 0; Col := 0; SumC := 0;
for i := 1 to 11 do
begin
  Temp := SumC + SumCol[i];
  Col := i;
  if (Temp >= MidArea) then break
  else SumC := Temp;
end;
if (MidArea - SumC) < (Temp - MidArea) then
  Col := Col - 1
else
  SumC := Temp;

// --------------------- end check column

// start select value from either sum-column or sum-row
if (Abs(SumC - MidArea) < Abs(SumR - MidArea)) then
begin
  X := Col;
  Result := 'C';
end else begin
  Y := Row;
  Result := 'R';
end;

procedure InitArray;
var i,j: integer;
begin
  for j := 1 to 11 do
    for i := 1 to 11 do
      Table[i,j] := Board[i,j];
  Area[0] := 0;
end;
for i := 1 to MaxTable do
begin
    arr1[i] := 0; arr2[i] := 0;
    arr3[i] := 0; arr4[i] := 0;
    Area[i] := RoomArea[i-1];
end;
for i := 1 to 4 do
begin
    Count[i] := 0; SumArea[i] := 0;
end;
for j := 1 to 11 do
    for i := 1 to 11 do
        if Table[i,j] <> 0 then
            arr1[table[i,j]] := Table[i,j];
    for i := 1 to 4 do
        count[i] := 0;
end;
Function TableArea(Table: array11_2D; x1,y1,x2,y2: integer): Integer;
Var
        i,j,Sum: Integer;
begin
    Sum := 0;
    for i := x1 to x2 do
        for j := y1 to y2 do
            Sum := Sum + Area[table[i,j]];
    Result := Sum;
end;
procedure FillArr(Var Arr: array20; x1,y1,x2,y2: integer;Table: array11_2D);
var
    i,j,value : integer;
begin
    for i := 1 to 20 do
        Arr[i] := 0;
    for j := y1 to y2 do
        for i := x1 to x2 do
            Value := Table[i,j];
            if (Value > 0) then
                arr[value] := Value;
end;
end;
Function CountTableX(Tbl: array11_2D; x1,y1,x2,y2,col: integer): Integer;
var
    i,j : integer;
begin
    Count1 := 0;
    for j := y1 to y2 do
        for i := x1 to Col do
            if tbl[i,j] > 0 then
                Inc(Count1);
    Count2 := 0;
    for j := y1 to y2 do
        for i := Col+1 to x2 do
            if tbl[i,j] > 0 then
                Inc(Count2);
end;
begin  
if tbl[i,j] > 0 then  
Inc(Count2);  
end;  
end;  
if (Count1 >= Count2) then Result := 1  
else Result := 2;  
end;  

Function CountTableY(Tbl : array1_2D; x1,y1,x2,y2,row : integer) : Integer;  
var i,j : integer;  
Count1,Count2 : integer;  
begin  
Count1 := 0;  
for j := y1 to Row do  
begin  
for i := x1 to x2 do  
begin  
if tbl[i,j] > 0 then  
Inc(Count1);  
end;  
end;  
Count2 := 0;  
for j := Row+1 to y2 do  
begin  
for i := x1 to x2 do  
begin  
if tbl[i,j] > 0 then  
Inc(Count2);  
end;  
end;  
if(Count1 >= Count2) then Result := 1  
else Result := 2;  
end;  

procedure TForm1.SplitTable(Sender : TObject;  
Var arr1,arr2 : array20;  
Var count1,count2 : integer;  
Var x11,y11,x12,y12 : integer;  
Var x21,y21,x22,y22 : integer);  
var pos : Char;  
all,x,y,xx,yy : integer;  
x1,y1,x2,y2 : integer;  
Percent : Extended;  
Table2 : array1_2D;  
begin  
  x1 := x11; x2 := x12; y1 := y11; y2 := y12;  
  TImage(Sender).Canvas.pen.color := clBlack;  
  TImage(Sender).Canvas.pen.width := 1;  
  for y := 1 to 11 do  
  begin  
    for x := 1 to 11 do  
    begin  
      table2[x,y] := 0;  
      all := table[x,y];  
      if arr1[all] <> 0 then  
      begin  
        table2[x,y] := all;  
      end;  
    end;  
  end;  
end;
all := TableArea(Table2,1,1,11,11);
pos := ScanTable(Table2,x,y);

if (pos="C") then begin
  y := TableArea(Table2,1,1,x,11)*100;
  Percent := y / all;
  FillArr(arr1,1,1,y,Table2);
  FillArr(arr2,x+1,1,11,11,Table2);
  xx := Round(percent*(x2-x1+1)) div 100;
  FillArr(arr1,1,1,11,11,Table2);
  FillArr(arr2,1,1,y,11,Table2);
  DrawRect(Sender,xl,yl,xx,y2);
  DrawRect(Sender,xl,yl,xx,y2);
  xl1 := xl; yl1 := yl; x12 := xx; y12 := y2;
x21 := xl; y21 := yl; x22 := x2; y22 := y2;
end else begin
  if (pos="R") then begin
    x := TableArea(Table2,1,1,y)*100;
    Percent := x / all;
    FillArr(arr1,1,1,11,11,Table2);
    FillArr(arr2,x+1,1,11,11,Table2);
    yy := Round(Percent*(y2-y1+1)) div 100;
    DrawRect(Sender,x1,y1,11,11,Table2);
    DrawRect(Sender,x1,y1,11,11,Table2);
    x11 := x1; y11 := y1; x12 := xx; y12 := y2;
x21 := xx; y21 := yl; x22 := x2; y22 := y2;
end end;

for x := 1 to MaxTable do begin
  if arr1[x] <> 0 then Inc(y);
end;
count1 := y;
for x := 1 to MaxTable do begin
  if arr2[x] <> 0 then Inc(y);
end;
count2 := y;

procedure WriteCoord(no,xl,yl,x2,y2 : integer);
begin
  bCoord[no].xl := Round(xl*Scale);
  bCoord[no].yl := Round(yl*Scale);
  bCoord[no].x2 := Round(x2*Scale);
  bCoord[no].y2 := Round(y2*Scale);
end;

function ScanArrayLeft(arr : array20) : integer;
var i : integer;
begin
  for i := 1 to MaxTable do
    if arr[i] > 0 then
procedure TForm1.FillColor(Sender : TObject; max : integer);
var i : integer;
midx,midy,size : integer;
begin
  TImage(Sender).Canvas.Font.Color := CLime;
  size := TImage(Sender).Canvas.Font.Size;
  for i := 1 to max do
  begin
    with bcoord[i] do
    begin
      if not PrintFlag then
      begin
        TImage(Sender).Canvas.Brush.Color := ColorTable[i];
        TImage(Sender).Canvas.FillRect(Rect(xl + 1,yl + 1,x2-1,y2-1));
      end;
      midx := (xl+x2) div 2;
      midy := (yl+y2) div 2;
      TImage(Sender).Canvas.TextOut(midx-Size,midy-Size,IntToStr(i));
    end;
  end;
end;

procedure TForm1.ShowDetail(Sender: TObject);
Var Row,Col,i: integer;
begin
  TImage(Sender).Canvas.Brush.Color := clWhite;
  Col := Round(100*Scale+20);
  for i := 1 to MaxTable do
  begin
    Row := 30*i;
    TImage(Sender).Canvas.Font.Color := ColorTable[i];
    TImage(Sender).Canvas.TextOut(Col,Row,Format('Area[%d] = %d',[i,Area[i]]));
  end;
end;

procedure TForm1.ShowLayout(Sender: TObject);
begin
  DrawRect(Sender,x1[1],y1[1],x2[1],y2[1]);
  initarray;
  TImage(Sender).picture := nil;
  Repeat
    SplitTable(Sender, arr1, arr2, count[1], count[2],
                 x1[1],y1[1],x2[1],y2[1],
                 x1[2],y1[2],x2[2],y2[2]);
    if (Count[2] > 1) then
      Repeat
        SplitTable(Sender, arr2, arr3, count[2], count[3],
                   x1[2],y1[2],x2[2],y2[2],
                   x1[3],y1[3],x2[3],y2[3]);
        if (Count[3] > 1) then
          Repeat
            SplitTable(Sender, arr3, arr4, count[3], count[4],
                       x1[3],y1[3],x2[3],y2[3],
                       x1[4],y1[4],x2[4],y2[4]);
            if (Count[4] > 1) then
              Repeat
end;
SplitTable(Sender, arr4, arr5, count[4], count[5],
    x1[4], y1[4], x2[4], y2[4],
    x1[5], y1[5], x2[5], y2[5]);
if (Count[5] > 1) then
    Repeat
        SplitTable(Sender, arr5, arr6, count[5], count[6],
            x1[5], y1[5], x2[5], y2[5],
            x1[6], y1[6], x2[6], y2[6]);
        if (Count[6] > 1) then
            Repeat
                SplitTable(Sender, arr6, arr7, count[6], count[7],
                    x1[6], y1[6], x2[6], y2[6],
                    x1[7], y1[7], x2[7], y2[7]);
                Application.ProcessMessages;
            Until Count[7] < 1;
        if (Count[6] = 1) then
            WriteCoord(ScanArrayLeft(arr6), x1[6], y1[6], x2[6], y2[6]);
        Application.ProcessMessages;
    Until Count[6] < 1;
if (Count[5] = 1) then begin
    WriteCoord(ScanArrayLeft(arr5), x1[5], y1[5], x2[5], y2[5]);
end;
Application.ProcessMessages;
Until Count[5] < 1;
if (Count[4] = 1) then begin
    WriteCoord(ScanArrayLeft(arr4), x1[4], y1[4], x2[4], y2[4]);
end;
Application.ProcessMessages;
Until Count[4] < 1;
if (Count[3] = 1) then begin
    WriteCoord(ScanArrayLeft(arr3), x1[3], y1[3], x2[3], y2[3]);
end;
Application.ProcessMessages;
Until Count[3] < 1;
if (Count[2] = 1) then begin
    WriteCoord(ScanArrayLeft(arr2), x1[2], y1[2], x2[2], y2[2]);
end;
Application.ProcessMessages;
Until Count[2] < 1;
WriteCoord(ScanArrayLeft(arr1), x1[1], y1[1], x2[1], y2[1]);

FillColor(Sender, MaxTable);
// ShowDetail(Sender);
end;

function CheckBorderX(i, j : integer) : Boolean;
begin
    if (bCoord[i].x1 = bCoord[j].x1) or
       (bCoord[i].x1 = bCoord[j].x2) or
       (bCoord[i].x2 = bCoord[j].x1) or
       (bCoord[i].x2 = bCoord[j].x2) then
    begin
        if ((bCoord[i].y1 in [bCoord[j].y1..bCoord[j].y2]) or
            (bCoord[i].y2 in [bCoord[j].y1..bCoord[j].y2]) or
            (bCoord[j].y1 in [bCoord[i].y1..bCoord[i].y2]) or
            (bCoord[j].y2 in [bCoord[i].y1..bCoord[i].y2])) then
            Result := True
        else
            Result := False;
    end
end
else
Result := False;
end;

function CheckBorderY(i,j : integer) : Boolean;
begin
if (bCoord[i].y1 = bCoord[j].y1) or
(bCoord[i].y1 = bCoord[j].y2) or
(bCoord[i].y2 = bCoord[j].y1) or
(bCoord[i].y2 = bCoord[j].y2) then
begin
if ((bCoord[i].x1 in ([bCoord[j].x1]..[bCoord[j].x2])) or
(bCoord[i].x2 in ([bCoord[j].x1]..[bCoord[j].x2])) or
(bCoord[j].x1 in ([bCoord[i].x1]..[bCoord[i].x2])) or
(bCoord[j].x2 in ([bCoord[i].x1]..[bCoord[i].x2]))) then
Result := True
else
Result := False;
end
else
Result := False;
end;

function CheckBorder(i,j : integer) : Boolean;
begin
if CheckBorderX(i,j) = True then
result := True
else begin
if CheckBorderY(i,j) = True then
Result := True
else
Result := False;
end;
end;

procedure CheckAdj(Var Tb2 : TTable);
var i,j : integer;
begin
for i := 1 to MaxTable-1 do
begin
for j := i+1 to MaxTable do
begin
(* if(i=1) and (j=2) then
begin
if CheckBorder(i,j) then
begin
tb2[i,j] := 1;
tb2[j,i] := 1;
end else
begin
tb2[i,j] := 0;
tb2[j,i] := 0;
end;
end else *)
if CheckBorder(i,j) then
begin
tb2[i,j] := 1;
tb2[j,i] := 1;
end else
begin

121
tb2[i,j] := 0;
tb2[j,i] := 0;
end;
end;
end;
end;

//-----------------------------------------------------------
procedure AdjTable(i,x,y: integer);
var Value : integer;
begin
// corner connect
Value := Board[x-1,y-1];
if Value in [1..20] then Adj[Value,i] := 0.7;
Value := Board[x+1,y-1];
if Value in [1..20] then Adj[Value,i] := 0.7;
Value := Board[x-1,y+1];
if Value in [1..20] then Adj[Value,i] := 0.7;
Value := Board[x+1,y+1];
if Value in [1..20] then Adj[Value,i] := 0.7;

// side connect
Value := Board[x-1,y];
if Value in [1..20] then Adj[Value,i] := 1;
Value := Board[x+1,y];
if Value in [1..20] then Adj[Value,i] := 1;
Value := Board[x,y-1];
if Value in [1..20] then Adj[Value,i] := 1;
Value := Board[x,y+1];
if Value in [1..20] then Adj[Value,i] := 1;
end;

procedure Adjustment;
var i,j: Integer;
begin
for i := 1 to 20 do // clear all adj.
begin
for j := 1 to 20
begin
Adj[i,j] := -0.0;
end;
end;

for j := 1 to 11 do
begin
for i:= 1 to 11 do
begin
if Board[i,j] in [1..20] then
begin
AdjTable(Board[i,j],i,j);
end;
end;
end;

procedure Distance;
var i,j : Integer;
begin
for j := 1 to MaxTable do
begin
  for i := 1 to MaxTable do
    begin
      if (i<>j) then
        Dist[i,j] := CalcDistance(i,j);
    end;
  end;

procedure CalcScore(n : integer);
var i,j : integer;
  Sum,SumTotal,SumRow : Extended;
begin
  SumToTal := 0.0;
  for j := 1 to MaxTable do
    begin
      SumRow := 0.0;
      for i := 1 to MaxTable do
        begin
          if (i <> j) then
            begin
              Sum := (LBTable[i,j]* Adj[i,j]);
              SumRow := SumRow + Sum;
            end;
        end;
      SumToTal := SumToTal + SumRow;
    end;
  Mark := SumToTal/2;
  if (SUMTCR=0) then SUMTCR:=1;
  case n of
    1 : Form1.lbScore1.Caption := Format('%0.2f,[Mark*100/SUMTCR]);
    2: Form1.lbScore2.Caption := Format('%0.2f,[Mark*100/SUMTCR]);
    3: Form1.lbScore3.Caption := Format('%0.2f,[Mark*100/SUMTCR]);
    4: Form1.lbScore4.Caption := Format('%0.2f,[Mark*100/SUMTCR]);
  end;
end;

procedure TForm1.DrawBoard(Sender:TObject);
var i,j,Value : Integer;
begin
  for j := 0 to MaxTable do
    begin
      for i := 0 to MaxTable do
        begin
          Value := Board[i+lj+l];
          if Value > 0 then
            TStringGrid(Sender).Cells[i,j] := IntToStr(Value)
          else
            TStringGrid(Sender).Cells[i,j] := ' ';
        end;
    end;
end;

procedure TForm1.Improve(Sender : TObject;no : integer);
var i,j,c,code,code2,icode,ocode,m,n,x,y,x2,y2,ox,oy: ShortInt;
  list : array[1..20] of Boolean;
  B : boardarray;
  rx,ry : array[1..20] of integer;
  MinMark : Extended;
  swap,noBetter : boolean;
begin
  for i := 1 to MaxTable do
    begin
      for j := 1 to MaxTable do
        begin
          Value := Board[i+lj+l];
          if Value > 0 then
            TStringGrid(Sender).Cells[i,j] := IntToStr(Value)
begin
  CalcTCR;
  noBetter := False;
  // ListBox3.Items.Clear;
  MinMark := Mark;
  Repeat
    swap := false;
    Adjustment;
    Distance;
    CalcScore(no);
  end;

  listBox2.items.Clear;
  MinMark := Mark;
(* case no of
  1 : lbScore1.Caption := Format('%0.2f',[MinMark]);
  2 : lbScore2.Caption := Format('%0.2f',[MinMark]);
  3 : lbScore3.Caption := Format('%0.2f',[MinMark]);
  4 : lbScore4.Caption := Format('%0.2f',[MinMark]);
end;*)

for i := 1 to 20 do List[i] := False;
for j := 1 to MaxTable do begin
  for i := 1 to MaxTable do begin
    if (i = j) then continue;
    if (LBTable[i,j] = 0) then continue;
    if (Adj[i,j] = 0.0) then begin
      List[i] := True; List[j] := True;
    end;
  end;
end;

for i := 1 to 20 do begin
  if List[i] then ListBox2.Items.Add(IntToStr(i));
end;

for i := 1 to 20 do begin
  rx[i] := roomx[i]; ry[i] := roomy[i];
end;

for m := 1 to 11 do // backup board
  for n := 1 to 11 do
    b[n,m] := board[n,m];

// drawboard;
// DiaGramPic1.Repaint;
// Application.ProcessMessages;
// ShowMessage('Click');

for c := 1 to ListBox2.Items.Count do begin
  (* for m := 1 to 20 do begin
    roomx[m] := rx[m];
    roomy[m] := ry[m];
  end;
  for m := 1 to 11 do
    for n := 1 to 11 do
      board[n,m] := b[n,m];
  end;*)
drawboard;
*)
code := StrToInt(ListBox2.Items[c-1]);
x := RoomX[code]; y := RoomY[code];

for j := 1 to 11 do begin
  for m := 1 to 20 do begin
    roomx[m] := rx[m];
    roomy[m] := ry[m];
  end;
  for m := 1 to 11 do // backup board
    for n := 1 to 11 do
      board[n,m] := b[n,m];
  // drawboard;

  // RoomX[code] := ox; RoomY[code] := oy;
  // Board[x,y] := code;
  for i := 1 to 11 do begin
    for m := 1 to 20 do begin
      roomx[m] := rx[m];
      roomy[m] := ry[m];
    end;
    for m := 1 to 11 do // backup board
      for n := 1 to 11 do
        board[n,m] := b[n,m];
    // drawboard;

    if (b[i,j] < 1) then begin
      RoomX[code] := i;
      RoomY[code] := j;
      Board[i,j] := Code;
      Board[x,y] := -1;
      code2 := 0; x2 := i; y2 := j;
    end else begin
      code2 := Board[i,j];
      x2 := RoomX[code2];
      y2 := RoomY[code2];
      RoomX[code] := x2;
      RoomY[code] := y2;
      Board[x2,y2] := code;
      RoomX[code2] := x;
      RoomY[code2] := y;
      Board[x,y] := code2;
    end;

  // DrawBoard;
  Adjustment;
  Distance;
  CalcScore(no);
  if (Mark > MinMark) then begin
    Listbox3.Items.Add(Format("%d,%d \rightarrow %0.4f, [%d,%d,%d,%d]");
    MinMark := Mark;
    Label4.Caption := Format("%0.4f", MinMark);
  end;
  //
ox := x2; oy := y2; ocode := code2;
icode := code; swap := true;
end else
begin if (Mark = MinMark) then
   // Listbox3.Items.Add(Format('%d,%d -> %0.4f,[code,code2,Mark/82]);
end;
Application.ProcessMessages;
end;
Application.ProcessMessages;
end;
end;
for m := 1 to 20 do
begin
   roomx[m] := rx[m];
   roomy[m] := ry[m];
end;
for m := 1 to 11 do
for n := 1 to 11 do
   board[n,m] := b[n,m];
if swap then
begin
   x := RoomX[icode]; y := RoomY[iCode];
   RoomX[icode] := ox; RoomY[iCode] := oy;
   Board[ox,oy] := icode;
   if (oCode<1) then begin
      Board[x,y] := -1;
   end
   else begin
      Board[x,y] := oCode;
      Roomx[oCode] := x;
      Roomy[oCode] := y;
   end;
end else begin
   // ShowMessage('No better .. !');
   NoBetter := True;
end;
// Listbox3.Items.Add('-------------- --');
case no of
1 : lbScore1.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
2 : lbScore2.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
3 : lbScore3.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
4 : lbScore4.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
end;
if lmproveCheck.Checked then DrawBoard(Sender);
Application.ProcessMessages;
U ltil noBetter;
Adjustment;
Distance;
CalcScore(no);
(* case no of
1 : lbScore1.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
2 : lbScore2.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
3 : lbScore3.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
4 : lbScore4.Caption := Format('%0.4f,[MinMark*100/SumTCR]);
end; *)
DrawBoard(Sender);
end;
procedure TForm1.ImproveBtnClick(Sender: TObject);
Var i : Integer;
FindLB : TComponent;
begin
  case ActiveDiagram of
    1 : begin
      ImportBoard(DiagramPic1);
      Improve(DiagramPic1,1);
      ShowLayout(LayoutPic1);
      CheckArrangeMent(1);
    end;
    2 : begin
      ImportBoard(DiagramPic2);
      Improve(DiagramPic2,2);
      ShowLayout(LayoutPic2);
      CheckArrangeMent(2);
      manual2.down := false;
    end;
    3 : begin
      ImportBoard(DiagramPic3);
      Improve(DiagramPic3,3);
      ShowLayout(LayoutPic3);
      CheckArrangeMent(3);
      manual3.down := false;
    end;
    4 : begin
      ImportBoard(DiagramPic4);
      Improve(DiagramPic4,4);
      ShowLayout(LayoutPic4);
      CheckArrangeMent(4);
      manual4.down := false;
    end;
  end;
  for i := 1 to 4 do
    begin
      FindLB := FindComponent('Manual'+ InttoStr(i));
      TSpeedButton(FindLB).Down := False;
      TSpeedButton(FindLB).Caption := 'Off';
      TSpeedButton(FindLB).Font.Color := clRed;
    end;

  ImproveBtn.Enabled := False;
end;

procedure TForm1.ImportBoard(Sender: TObject);
Var i,j, Value : Integer;
begin
  for j := 0 to 10 do
    begin
      for i := 0 to 10 do
        begin
          Value := StrToIntDef(TStringGrid(Sender).Cells[i,j],-1);
          Board[i+1,j+1] := Value;
          if Value > 0 then
            begin
              Roomx[Value] := i+1;
              Roomy[Value] := j+1;
            end;
        end;
    end;
  DrawBoard(Sender);
end;
procedure TForm1.Manual1Click(Sender: TObject);
var i : Integer;
   FindLB : TComponent;
begin
  If TSpeedButton(Sender).Down then
    begin
      ActiveDiagram := TSpeedButton(Sender).Tag;
      ImproveBtn.Enabled := True;
    end
  else
    begin
      ActiveDiagram := 0;
      ImproveBtn.Enabled := False;
    end;
  for i:= 1 to 4 do
    begin
      FindLB := FindComponent('Manual'+ IntToStr(i));
      if TSpeedButton(FindLB).Down then
        begin
          TSpeedButton(FindLB).Caption := 'On';
          TSpeedButton(FindLB).Font.Color := clGreen;
        end
      else
        begin
          TSpeedButton(FindLB).Caption := 'Off';
          TSpeedButton(FindLB).Font.Color := clRed;
        end;
    end;
  case ActiveDiagram of
    1 : ImportBoard(DiaGramPic1);
    2 : ImportBoard(DiaGramPic2);
    3 : ImportBoard(DiaGramPic3);
    4 : ImportBoard(DiaGramPic4);
  end;
end;

procedure TForm1.EvalTableDrawCell(Sender: TObject; vCol, vRow: LongInt;
  Rect: TRect; State: TGridDrawState);
var x : integer;
begin
  if Sender = ActiveControl then exit;
  if not (gdSelected in State) then exit;
  with Sender as TStringGrid do
    begin
      case vCol of
        0..4: begin
          SetTextAlign(Canvas.Handle, TA_CENTER);
          x := (Rect.Left+Rect.Right) div 2;
        end;
        else begin
          SetTextAlign(Canvas.Handle, TA_LEFT);
          x := Rect.Left+2;
        end;
      end;
      case vRow of
        0: begin
          SetTextAlign(Canvas.Handle, TA_CENTER);
          x := (Rect.Left+Rect.Right) div 2;
        end;
      end;
    end;
end;
if not (gdFixed in state) then
begin
  Canvas.Brush.Color := Color;
  Canvas.Font := Font;
end;
Canvas.TextRect(Rect,x,Rect.Top+2,Cells[vCol,vRow]);
end;

procedure TForm1.Config1Click(Sender: TObject);
begin
  Form4.Showmodal;
  CalcTCR;
end;

procedure TForm1.DiagramPic1DrawCell(Sender: TObject; vCol, vRow: Integer;
  Rect: TRect; State: TGridDrawState);
begin
  if Sender = Activecontrol then Exit;
  with Sender as TStringGrid do
  begin
    canvas.brush.color := color;
    canvas.font := font;
    canvas.textrect(rect,rect.left+2,rect.top+2,cells[vcol,vrow]);
  end;
end;

procedure TForm1.DiagramPic1MouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  if TStringGrid(Sender).Tag = ActiveDiagram then
    with Sender as TStringGrid do { treat Sender as TFileListBox }
    begin
      MoveText := StrToIntDef(TStringGrid(Sender).Cells[SelCol,SelRow],0);
      MoveText := Board[SelCol+1,SelRow+1];
      OldCol := SelCol;
      OldRow := SelRow;
    end;
end;

procedure TForm1.DiagramPic1MouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if TStringGrid(Sender).Tag = ActiveDiagram then
    with Sender as TStringGrid do { treat Sender as TFileListBox }
    begin
      if Board[SelCol+1,SelRow+1] < 1 then
        begin
          Board[OldCol+1,OldRow+1] := -1;
          Board[SelCol+1,SelRow+1] := MoveText;
          RoomX[MoveText] := SelCol+1;
          RoomY[MoveText] := SelRow+1;
          MoveText := -1;
          DrawBoard(Sender);
          case ActiveDiagram of
            1: ShowLayout(LayOutPic1);
            2: ShowLayout(LayOutPic2);
            3: ShowLayout(LayOutPic3);
            4: ShowLayout(LayOutPic4);
          end;
        end;
    end;
end;
procedure TForm1.DiagramPic1SelectCell(Sender: TObject; Col, Row: Integer; var CanSelect: Boolean);
begin
SelCol := Col;
SelRow := Row;
end;

procedure TForm1.Step4Click(Sender: TObject);
begin
NoteBook1.PageIndex := TSpeedButton(Sender).Tag;
end;

procedure TForm1.CheckArrangement(Col: integer);
Var
Th2 : TTable;
iJ,Value : integer;
Sum, SumA,SumE,SumI,SumO,SumU,SumX : Integer;
begin
SumA := 0; SumE := 0; SumI := 0;
SumO := 0; SumU := 0; SumX := 0;
CheckAdj(tb2);
Form6.StringGrid1.ColCount := MaxTable+1;
Form6.StringGrid1.RowCount := MaxTable+1;
Sum := 0;
for j := 1 to MaxTable do
begin
for i := j+1 to MaxTable do
begin
CheckTBl[Col][i][j] := 0;
Value := LbTable[i,j];
if(tb2[i,j]=1) then
begin
CheckTBl[Col][i][j] := 1;
Sum := Sum + Value;
case Value of
 4 : Inc(SumA);
 3 : Inc(SumE);
 2 : Inc(SumI);
 1 : Inc(SumO);
 0 : Inc(SumU);
-1 : Inc(SumX);
end;
end;
end;
i :=Col;
With EvalTable do
begin
Cells[i,1] := IntToStr(SumA);
Cells[i,2] := IntToStr(SumE);
Cells[i,3] := IntToStr(SumI);
Cells[i,4] := IntToStr(SumO);
end;
procedure TForm1.Button2Click(Sender: TObject);
var i, j: integer;
begin
  Form6.check := TButton(Sender).Tag;
  for j := 0 to maxtable do
    begin
      Form6.StringGrid1.Cells[0,j] := IntToStr(j);
      Form6.StringGrid1.Cells[j,0] := IntToStr(j);
    end;
  for j := 1 to maxtable do
    begin
      for i := j+1 to maxtable do
        begin
          Form6.StringGrid1.Cells[i,j] := StChar[LBTable[i,j]]
        end;
    end;
  Form6.ShowModal;
end;

procedure SaveAreaToFile(FileName : String; No : Integer);
Var myini : TIniFile;
  i: integer;
  Ident : String;
begin
  myini := TIniFile.Create(FileName);
  for i := 0 to No-1 do
    begin
      Ident := 'Area'+IntToStr(i);
      myini.WriteLine('Area',Ident,RoomArea[i] div 10);
    end;
  myini.free;
end;

procedure SaveFunctionToFile(FileName: String; No : Integer);
var myini : TIniFile;
i: integer;
Ident : String;
begin
  myini := TIniFile.Create(FileName);
  for i := 0 to No-1 do
    begin
      Ident := 'F'+IntToStr(i);
      myini.WriteLine('Function',Ident,Func[i]);
    end;
  myini.free;
end;

procedure SaveNameToFile(FileName : String; No: Integer);
var myini : TIniFile;
i: integer;
Ident : String;
begin
  myini := TIniFile.Create(FileName);
  for i := 0 to No-1 do
    begin
      Ident := 'N'+IntToStr(i);
      myini.WriteLine('Name',Ident,Name[i]);
    end;
  myini.free;
end;
begin
  myini := TIniFile.Create(FileName);
  for i := 0 to No-1 do
  begin
    Ident := 'Name'+IntToStr(i);
    myini.WriteString('Name',Ident,RoomName[i]);
  end;
  myini.free;
end;

procedure SaveReasonToFile(FileName: String; MaxTable: Integer);
var myini : TIniFile;
i : integer;
Ident : String;
begin
  myini := TIniFile.Create(FileName);
  for i := 0 to MaxTable do
  begin
    Ident := 'Reason '+IntToStr(i);
    myini.WriteString('Reason',Ident,RoomSt[i]);
  end;
  myini.free;
end;

procedure TForm1.Save1Click(Sender: TObject);
var txt : textfile;
filename : String;
i,j : integer;
begin
  if not savedialog.Execute then
    exit;
  filename:= SaveDialog1.filename;
  AssignFile( txt,filename );
  {1-} rewrite(txt); {1+}
  if (IOResult <> 0) then
    begin
      ShowMessage('Error: Can’t Save Data File!');
      exit;
    end;
  WriteLn( txt, 'dept=',maxtable);
  for j := 1 to maxtable do
    begin
      for i := 1 to maxtable do
        begin
          if (i=j) then
            Write(txt,'@')
          else
            Write(txt,StChar[LBTable[i,j]]);
          if i < maxtable then
            Write( txt,' ,');
        end;
      WriteLn(txt);
    end;
  WriteLn(txt);
  for j := 1 to maxtable do
    begin
      if (i=j) then
        Write(txt,'@')
      else
        Write(txt,StChar[LBTable[i,j]]);
      if i < maxtable then
        Write(txt,' ,');
    end;
  WriteLn(txt);
end;
for j := 1 to maxtable do
begin
WriteLn(txt);
end;
for i := 1 to maxtable do 
begin 
if (i=j) then
  Write(txt,'@') 
else 
  Write(txt,RSTable[i,j]);
if i < maxtable then 
  Write(txt,',');
end;
WriteLn(txt);
end;
CloseFile(txt);
SaveReasonToFile(FileName,MaxTable);
SaveNameToFile(FileName,MaxTable);
SaveAreaToFile(FileName,MaxTable);
SaveFunctionToFile(FileName,MaxTable);
end;

procedure TForm1.ClearAll;
var i,j : integer;
begin 
  lbScore1.Caption := 'Score!';
  lbScore2.Caption := 'Score2';
  lbScore3.Caption := 'Score3';
  lbScore4.Caption := 'Score4';
  lbTime1.Caption := 'Time!';
  lbTime2.Caption := 'Time2';
  lbTime3.Caption := 'Time3';
  lbTime4.Caption := 'Time4';
  ClearRoom(DiagramPic1);
  ClearRoom(DiagramPic2);
  ClearRoom(DiagramPic3);
  ClearRoom(DiagramPic4);
  LayoutPic1.Picture := Nil;
  LayoutPic2.Picture := Nil;
  LayoutPic3.Picture := Nil;
  LayoutPic4.Picture := Nil;
for j := 1 to 7 do 
begin 
  for i := 1 to 4 do 
begin 
    EvalTable.Cells[i,j] := "
  end;
end;
for j := 0 to 20 do 
begin 
  for i := 0 to 20 do 
begin 
    LBTable[i,j] := 0;
  end;
end;
ColorCheckBox.Checked := True;
NewData := True;
end;

procedure TForm1.TCRTable1Click(Sender: TObject);
begin 
  Form7.ShowModal;
end;
end;

procedure TForm1.ColorCheckBoxClick(Sender: TObject);
var FindLB : TComponent;
i,Col,Row: integer;
begin
  For i := 1 to LBCount do {LBCount 1 to 190}
  begin
    FindLB := FindComponent('LB'+ IntToStr(i));
    TLabel(FindLB).Visible := True;
    NumToCor(i,Col,Row);
    if ColorCheckBox.Checked then
      TLabel(FindLB).Font.Color := FontColor[LBTable[Col,Row]]
    else
      TLabel(FindLB).Font.Color := ClWhite;
    Application.ProcessMessages;
  end;
end;

procedure TForm1.PrintLayout(no: integer);
Var Score : String;
begin
  case No of
  1 : ImportBoard(DiaGramPic1);
  2 : ImportBoard(DiaGramPic2);
  3 : ImportBoard(DiaGramPic3);
  4: ImportBoard(DiaGramPic4);
  end;
  Scale := 5;
  Form1.ShowLayout(Form5.Lrmaget);
  Score:= EvalTable.Cells[No,7];
  Form5.lbScore.Caption := 'Score= '+Score;
  Form5.lbScore.Visible := True;
  Form5.BorderStyle := bsNone;
  Form5.Color := clWindow;
  Form5.BitBtn1.Visible := False;
  // Form5.Show:
  Form5.Print;
  Form5.BitBtn1.Visible := True;
  Form5.BorderStyle := bsToolWindow;
  Scale := 2.5;
  Form5.lbScore.Visible := False;
end;

procedure TForm1.Print1Click(Sender: TObject);
begin
  if (PrintLayoutDlg.ShowModal = mrOK) then begin
    PrintFlag := True;
    if PrintLayoutDlg.CheckBox1.Checked then
      PrintRolChart(Sender);
    if PrintLayoutDlg.CheckBox2.Checked then
      PrintLayout1;
    if PrintLayoutDlg.CheckBox3.Checked then
      PrintLayout2;
    if PrintLayoutDlg.CheckBox4.Checked then
      PrintLayout3;
    if PrintLayoutDlg.CheckBox5.Checked then
      PrintLayout4;
  end;
PrintFlag := False;
end;

procedure TForm1.PrintRolChart(Sender: TObject);
Var i,Col,Row,Value: Integer;
FindLB, FindIm : TComponent;
aBitmap : TBitmap;
begin
  with Form8 do
  begin
    for i := 1 to 190 do
      begin
        FindLB := FindComponent('LB'+ IntToStr(i));
        if i <= LBCount then
          begin
            TLabel(FindLB).Visible := True;
            NumToCor(i,Col,Row);
            Value := LBTable[Col,Row];
            TLabel(FindLB).Caption := StChar[Value];
            TLabel(FindLB).Font.Color := CIBlack;
          end else
          TLabel(FindLB).Visible := False;
      end;

    for i := 0 to ListView1.Items.Count-1 do
      begin
        img[i+1] := ListView1.Items[i].ImageIndex;
        Func[i] := img[i+1];
        RoomName[i] := ListView1.Items[i].Caption;
        RoomArea[i] := StrToInt(ListView1.Items[i].SubItems[0]);
      end;

    for i := 1 to 20 do {Clear all}
      begin
        FindIm := FindComponent('Sym'+ IntToStr(i)) as TImage;
        ImageList1.GetBitmap(img[i],aBitmap);
        Timage(FindIm).picture.assign(aBitmap);
        Timage(FindIm).Visible := True;
        FindLB := FindComponent('ActName'+ IntToStr(i));
        TLabel(FindLB).Caption := IntToStr(i) +': '+ ListView1.Items[i-1].Caption;
        TLabel(FindLB).Visible := True;
        aBitmap.Free;
      end;

    for i := 1 to ActNum do {ActNum 1 to 20}
      begin
        aBitmap := TBitmap.Create;
        FindIm := FindComponent('Sym'+ IntToStr(i));
        ImageList1.GetBitmap(img[i],aBitmap);
        Timage(FindIm).picture.assign(aBitmap);
        Timage(FindIm).Visible := True;
        FindLB := FindComponent('ActName'+ IntToStr(i));
        TLabel(FindLB).Caption := IntToStr(i) +': ' + ListView1.Items[i-1].Caption; {Show ActName in RelChart}
        TLabel(FindLB).Visible := True;
        aBitmap.Free;
    end;

    Form8.Show;
    Form8.Print;
  end;
procedure TForm1.PrintSetup1Click(Sender: TObject);
begin
  PrinterSetUpDialog1.Execute;
end;

procedure TForm1.New1Click(Sender: TObject);
begin
  ListView1.Items.Clear;
  Step1.Enabled := True;
  Step2.Enabled := False;
  Step3.Enabled := False;
  Step4.Enabled := False;
  Step5.Enabled := False;
  EstabView.Enabled := True;
  ChartView.Enabled := False;
  DiaView.Enabled := False;
  DrawView.Enabled := False;
  EvalView.Enabled := False;
  TCRTable1.Enabled := False;
  ClearAll;
  NoteBook1.PageIndex := 0;
end;

procedure TForm1.CheckBox2Click(Sender: TObject);
begin
  if CheckBox2.Checked then
    Form2.Show
  else
    Form2.Close;
    Form1.SetFocus;
end;

procedure TForm1.ListView1DblClick(Sender: TObject);
var Index : Integer;
aBitmap : TBitmap;
begin
  if ListView1.Items.Count <> 0 then
  begin
    Index := ListView1.ItemFocused.Index;
    NameIn.Text := ListView1.Items[Index].Caption;
    AreaIn.Text := ListView1.Items[Index].SubItems[0];
    FunctionIn.ItemIndex := ListView1.Items[Index].ImageIndex;
    aBitmap := TBitmap.Create;
    ImageList2.GetBitmap(FunctionIn.ItemIndex,aBitmap);
    ImPreview.picture.assign(aBitmap);
    aBitmap.Free;
  end;
end;

procedure TForm1.UpdBtnClick(Sender: TObject);
var Index : Integer;
begin
  UpdBtn.Enabled := True;
  UpdBtn.Default := True;
  AddBtn.Enabled := False;
  InsBtn.Enabled := False;
  DelBtn.Enabled := False;
  ClrBtn.Enabled := False;
  ListView1.Enabled := False;
end;

procedure TForm1.UpdBmClick(Sender: TObject);
var Index : Integer;
begin
If ListView1.Items.Count <> 0 then
begin
    Index := ListView1.ItemFocused.Index;
    ListView1.Items[Index].Caption := NameIn.Text;
    ListView1.Items[Index].SubItems[0] := AreaIn.Text;
    ListView1.Items[Index].ImageIndex := FunctionIn.ItemIndex; {Change ImPreview when add}
    UpdBtn.Enabled := False;
    AddBtn.Enabled := True;
    InsBtn.Enabled := True;
    DelBtn.Enabled := True;
    ClrBtn.Enabled := True;
    ListView1.Enabled := True;
    NameIn.Clear;
    AreaIn.Clear;
    NameIn.SetFocus;
end;
end;

procedure TForm1.SavePicture1Click(Sender: TObject);
Var FileName : String;
    ImageTag : Integer;
begin
    ImageTag := TPopupMenu(Sender).Tag;
    ImgSaveDialog.FileName := 'Layout'+IntToStr(ImageTag);
    if ImgSaveDialog.Execute then
    begin
        FileName := ImgSaveDialog.FileName;
        case ImageTag of
            1 : Layoutpic1.Picture.SaveToFile(FileName);
            2 : Layoutpic2.Picture.SaveToFile(FileName);
            3 : Layoutpic3.Picture.SaveToFile(FileName);
            4 : Layoutpic4.Picture.SaveToFile(FileName);
        end;
    end;
end;

procedure TForm1.ZoomPicture1Click(Sender: TObject);
Var ImageTag : Integer;
begin
    ImageTag := TPopupMenu(Sender).Tag;
    case ImageTag of
        1: ImportBoard(DiaGramPic1);
        2: ImportBoard(DiaGramPic2);
        3: ImportBoard(DiaGramPic3);
        4: ImportBoard(DiaGramPic4);
    end;
    Scale := 5;
    Form1.ShowLayout(Form5.Image1);
    Form5.ShowModal;
    Scale := 2.5;
end;

procedure TForm1.EstabViewClick(Sender: TObject);
begin
    Step1.Click;
end;

procedure TForm1.ChartViewClick(Sender: TObject);
begin
Step2.Click;
end;

procedure TForm1.DiaViewClick(Sender: TObject);
begin
  Step3.Click;
end;

procedure TForm1.DrawViewClick(Sender: TObject);
begin
  Step4.Click;
end;

procedure TForm1.EvalViewClick(Sender: TObject);
begin
  Step5.Click;
end;

procedure TForm1.About1Click(Sender: TObject);
begin
  AboutBox.Showmodal;
end;

end.
BIBLIOGRAPHY


