ED 084 881 EM 011 705

AUTHOR Mitchell, Ron; Conner Michael
TITLE A Brief Description of the Purposes and Concepts of the Coursewriter II Preprocessor. System Memo Number Four.
INSTITUTION Texas Univ., Austin. Computer-Assisted Instruction Lab.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO SM-4
PUB DATE May 71
NOTE 7p.

EDRS PRICE MF-$0.65 HC-$3.29
DESCRIPTORS *Computer Assisted Instruction; *Computer Programs; Curriculum Development; *Laboratory Manuals; Programing Languages
IDENTIFIERS *Coursewriter II Preprocessor

ABSTRACT

A brief description of the Coursewriter II preprocessor is provided. This preprocessor, a program written in FORTRAN IV on the CDC 6600 computer, is designed to reduce the repetition of effort that takes place from the time of the author's conception of a course to the time of its availability for on-line student instruction. The programer deals mainly with two types of information: 1) control logic or course structure information, and 2) content information. The preprocessor enables the programer to deal with the latter of these in a more natural way than Coursewriter II allows. The effectiveness of the preprocessor is currently being evaluated by using it to implement a 1500 Coursewriter II precalculus math course. (CH)

THE UNIVERSITY OF TEXAS AT AUSTIN

Computer Assisted Instruction Laboratory

AUSTIN

EM 011 705

A BRIEF DESCRIPTION OF THE PURPOSES AND CONCEPTS OF THE COURSEWRITER II PREPROCESSOR

SYSTEMS MEMO NO. 4

Ron Mitchell
Mike Conner

May 1971

Supported By:

THE NATIONAL SCIENCE FOUNDATION
Grant GJ 509 X

Computer-Assisted Instruction Laboratory
C. Victor Bunderson, Director
The University of Texas at Austin
Austin, Texas 78712

A BRIEF DESCRIPTION OF THE PURPOSES AND CONCEPTS OF THE COURSEWRITER II PREPROCESSOR

The Coursewriter II preprocessor is a program written in FORTRAN IV on the CDC 6600 computer. The output is Coursewriter II statements on cards, ready for input to the 1500 Coursewriter II card assembler. The 6600 was used because of the more powerful FORTRAN compiler and because of the extremely fast turn-around. A more extensive description of the preprocessor may be found in *A User's Guide to the Preprocessor.*[1] This document provides too much detail and has too specific an application for the personnel and procedures at The University of Texas Computer-Assisted Instruction Laboratory to be of general interest. Therefore, this short description of the preprocessor was provided to serve a more general audience. The effectiveness of the preprocessor is currently being evaluated by using it to implement a 1500 C rsewriter II precalculus math course. The results of this evaluation will be published at a later date as a systems memo, available at The University of Texas Computer-Assisted Instruction Laboratory, Austin, Texas.

[1]Ron Mitchell and Mike Conner. *A User's Guide to the Preprocessor.* Systems Memo No. 3, Computer-Assisted Instruction Laboratory, The University of Texas, Austin, Texas, 1971.

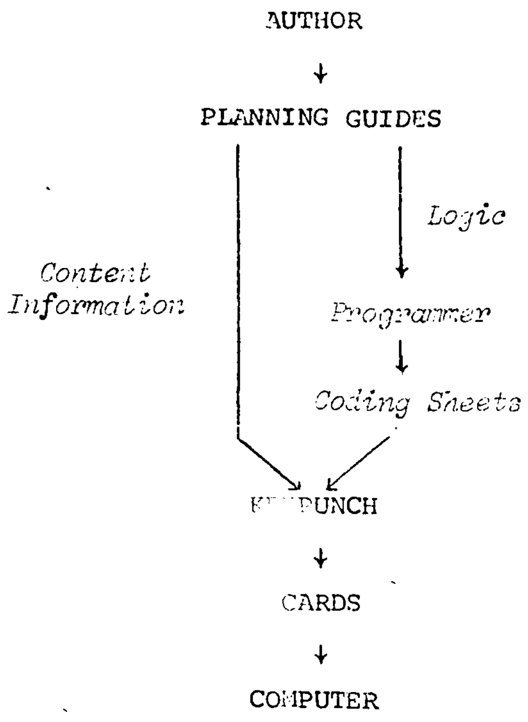
The preprocessor was designed to reduce the repetition of effort which takes place from the time of the author's conception of a course to the time of its availability for on-line student instruction. The process of implementation is normally stepwise, as shown below:

$$\begin{tikzcd}
\text{AUTHOR} \arrow[d] \\
\text{PLANNING GUIDES} \arrow[d] \\
\text{PROGRAMMER} \arrow[d] \\
\text{CODING SHEETS} \arrow[d] \\
\text{KEYPUNCH} \arrow[d] \\
\text{CARDS} \arrow[d] \\
\text{COMPUTER}
\end{tikzcd}$$

The programmer deals mainly with two types of information: (1) control logic or course structure information, and (2) content information. The preprocessor was designed to enable the programmer to deal with the latter of these in a more natural way than Coursewriter II (CW II) allows.

In most cases, the course content information is completely and explicitly stated in the author's planning guides, and the task of the programmer is, for the most part, that of the tedious and completely mechanical translation of the planning guides into the appropriate sequence of CW II statements. It is difficult, if not impossible, to perform this tedious translation on even small amounts of material, much less on the enormous amounts required by a major course, without introducing a tremendous number of errors.

The solution to this problem is to make use of that content information which is explicitly stated in the planning guides which would thus relieve the programmer as much as possible of this mechanical task and enable him to concentrate on the program logic. The steps of progressing from course conception to implementation would be the following:



In making this bypass, the following constraints must be observed:

(1) Keypunching must remain more or less a transcription task.

(2) No additional restrictions must be placed on the author.

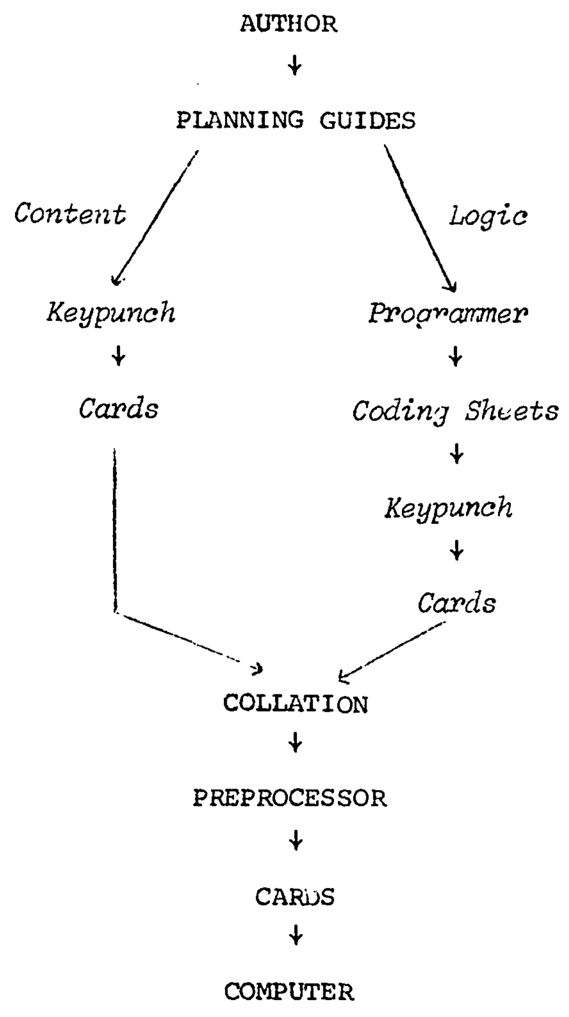
(3) A means for collation of the transcribed course content information and the programmer-produced course logic must be provided.

(4) The programming of the logic should be made easier and clearer by allowing the programmer a convenient and natural (in relation to programming) means of referencing the course content.

The preprocessor was designed to make this bypass according to the above criteria.

The preprocessor has added seven op-codes to the CW II language. These op-codes allow course content information to be entered directly from the planning guides by the keypunch operator and to be referenced within the flow of course logic by the programmer.

The course content information which is explicitly stated in the planning guides and recognized by the preprocessor is of two types: message data and variable data. Message data are display templates which are constant except for references to variable data. Variable data consist of variable length literal strings which may be referenced by the programmer. With the preprocessor, the flow from author to computer would be as shown in the following diagram:



The keypunch operator can define message data and variable data from the planning guides (with a minimal amount of effort by the programmer) by use of certain of the op-codes, and the programmer has easy access to these data by variable reference and by use of another of the op-codes. Collation has been reduced to a trivial task since the information transcribed by the keypunch operator is analogous to declarations in ALGOL and appears before any program logic produced by the programmer. Brief descriptions of the added op-codes are shown below:

BEGIN PROBLEM (BP). . . .	Signifies the beginning of a problem and defines the base-label for the problem.
VARIABLE DECLARATION (VD) .	Allows variable data to be declared and variable attributes to be assigned.
ENTER DATA (ED)	Enables a table of variable values to be created.
MESSAGE DEFINITION (MD).	Allows message data to be defined and assigned labels by which they may later be referenced. Message data may in turn contain references to variable data.
DISPLAY MESSAGE (DM). . .	Enables the programmer to reference, by label, previously defined message data.
PUNCH TABLE (PT).	Causes generation of a series of statements that provide the means by which variable data take on new values.
END PROBLEM (ZZ).	Signals the end of a problem and causes reinitialization of all pointers, flags, and tables.