

# **BUS RESERVATION SYSTEM:**

## **PROJECT REPORT**

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT  
FOR THE AWARD OF THE DEGREE OF

## **BACHELOR OF TECHNOLOGY**

(Branch :- Computer Science and Engineering )

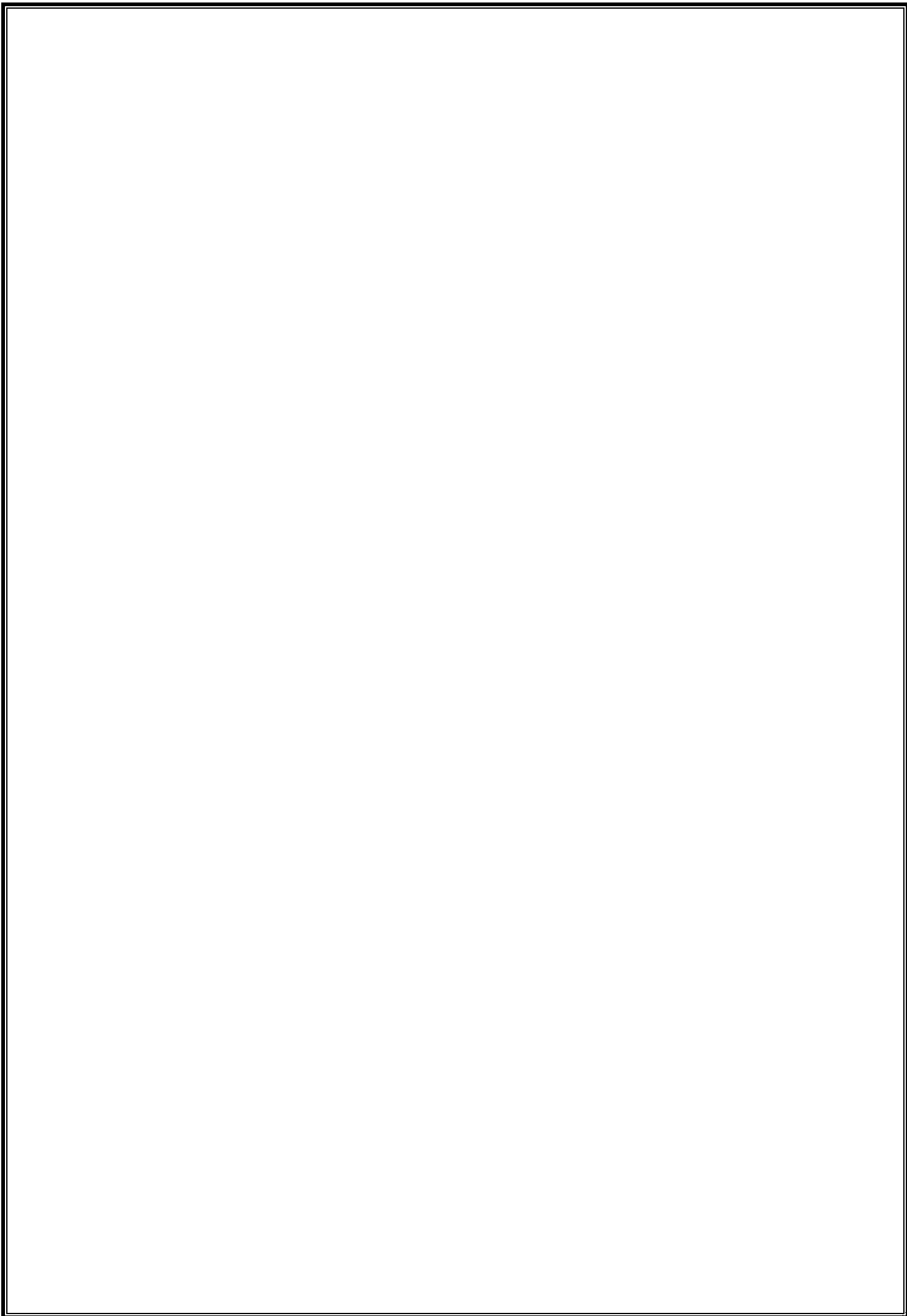
SUBMITTED BY

(RISHABH KUMAR)

(2200460100088)



**Dr. APJ Abdul Kalam Technical University, Uttar Pradesh**  
LUCKNOW, INDIA



## INTRODUCTION

Are you looking for an online bus booking system project? We are here to help you. You can [contact us](#). Online Bus Booking System cloud-based online software. This system would help customers to book a seat for their journey, book a bus. This system would also help the owner to manage the coaches, employees, clients, services, etc.

Bus Reservation System will increase the booking process faster, convenient, and comfortable. Customers can book their desired seats. They can check the availability of posts on a specific date. The customer can check availability, book a ticket, or cancel a ticket 24X7. The online system is available to use anytime. The user doesn't require to visit any office.

They just need internet and device to use our system. They can check the route, price, class, etc. They can pay the fare using a credit card, debit card, internet banking, online wallet like Paytm, and cash too. Managing buses, employees, and salaries would be very comfortable using this system. This is a safe and secure way to expand the business.

The system decreases human efforts and increases customer satisfaction.

▪

### **Modules of Online Bus Booking System:**

There are several modules required to complete this system. Here we are discussing the main modules or core modules of the system.

## **OBJECTIVE**

The main purpose of this study is to automate the manual procedures of reserving a bus ticket for any journey made through Imo Transport Company (ITC).

this system is said to be an automatic system and customers can select seats by themselves. Specifically, objectives of this project will consist of:

- i) Providing a web-based bus ticket reservation function where a customer can buy bus ticket through the online system without a need to queue up at the counter to purchase a bus ticket.
- ii) Enabling customers to check the availability and types of busses online. Customer can check the time departure for every ITC bus through the system.
- iii) Easing bus ticket payment by obtaining a bank pin after payments is made to the various designated banks.
- iv) Ability of customers to cancel their reservation.
- v) Admin user privileges in updating and canceling payment, route and vehicle records.

## SOFTWARE REQUIREMENTS

Code::html

Visual studio code or notepad

## HARDWARE REQUIREMENTS

**Operating system:** Windows 7 And Later

**Processor:** Intel corei3 and later

**Disk Space:** 1GB

**Memory:** 512 MB RAM

## FLOWCHART

level of DFD. Figures 3.1 to 3.3 show a diagram of the flow of data about the system

### Level 0

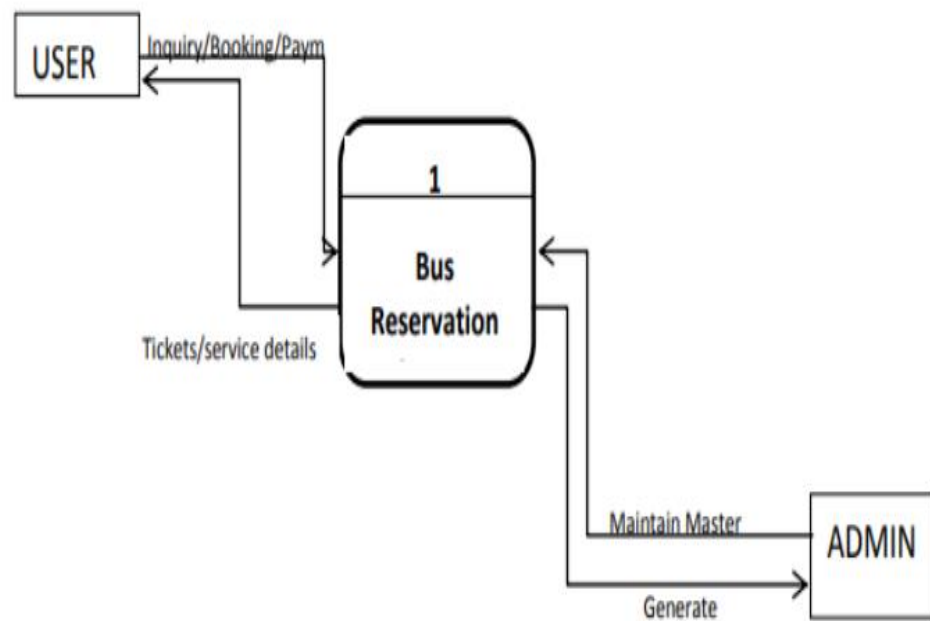


Figure 3.2 Context View of Online Bus Ticket Reservation System

## LEVEL 1

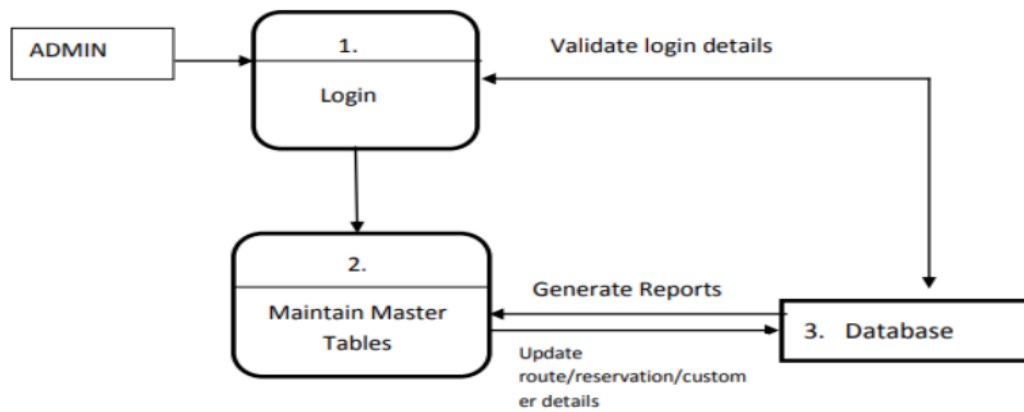


Figure 3.3 User view of Online Bus Ticket Reservation System

## LITERATURE REVIEW

Over the years mobile payments have been analyzed under various occasions and circumstances based on the findings each author wanted to present at the particular time of writing.

The literature review is based on different studies, conference papers, dissertations and various other relevant resources that would be very beneficial for designing and writing my dissertation proposal.

Accenture is a worldwide multinational business, which has as an area of expertise the management consulting services. They published a very interesting paper, which aims to determine and measure the capabilities of Mobile Ticketing technology in the area of public transportation.

More 9 precise they aim to identify the convenience and efficiency of such pioneer implementations, in today's modern world. Just like they mention in their publication, the advanced mobile technology that exists nowadays "enable citizens to purchase and carry electronic tickets for public transportation using their mobile phones, at the exact place and time at which transportation is needed" Accenture (no date).

However the author approaches the difficulties of adopting such a solution, due to three main factors: 1) Return of investment in tax payers' 2) the technology that will be used in these systems might be outdated soon (so this means more technology-related expenses in order to upgrade the equipment) 3) commuters might seek for multiple ways to avoid paying the ticket. Furthermore they present a case study they conduct for the Finnish Rail company, where they install a wireless ticket sales system in order to realize if this solution will improve the customer experience.

The conclusion of this paper clearly states that there are various opportunities inherent in the industry of mobile ticketing; however there are some real dangers that should be taken in consideration, like the return of investment and the fact that the solution must provide efficiency and flexibility.

Closing the approach of this paper I can say that it has been very beneficial for my research and me, due to the fact that it revealed potential threats that I should eliminate on my implementation.

A very similar paper with my dissertation proposal comes with Anguranak et al.(2015) who make use of same type of smart phone communication technology (NFC), in order to validate and pay for some kind of goods or services.

In that particular case study the author has created a smart phone and NFC communication system, where the users could pay the amount that required in order to buy printing credits for the computer lab. In his paper the author describes in a very accurate way the technologies he used and the entire database structure for the sake of his solution.

He chose to create a Java application in order to render and handle the communication between the smart phone and the database. Anguranak et al. (2015) instead of forcing the user to create a user account on his application, he preferred to concentrate on the Student ID number that is been used as a verification ID number.



Having in mind all the above and in co-existence with the valuable paper of Zhuo-yi et al.

(2012) who presented a portable ticket validating system based on ARM and GPRS technology, I was capable to fully comprehend in depth on what technologically direction to rely on.

The above two researches provide me with useful insights about the database structure and in addition they made me re-evaluate the original thoughts I had for the design and creation of the database and the entire ticket validating procedure in my proposal.

On the other hand of Accenture (no date) ambitious attitude we find the Apansevic et al. (2014), which is commenting about one of the major areas of the mobile payment application in Sweden, which is the mobile public transport ticketing

. In more depth they examine this trend from the stakeholder's point of view, where the majority of them are -as expected- mobile network operators. This paper is presenting us a new mobile payment method called "MyWallet", which was introduced in Sweden at the early of 2013.

Its primary focus has to do with the expectations that weren't occurred for this new but yet promising SMS payment service.

This paper has been very helpful for my dissertation progress, because it exposed the vulnerabilities and the drawbacks that have been appeared in the solution that took place in Sweden.

More precise issues like price difference policies between transportation companies, the diversity of the ticketing systems and in total the lack of common system cooperation, concluded to characterize this system as a major disaster.

It is really interesting how Apansevic et al. (2014) and Accenture (no date), conclude in complete opposite findings for almost the same solution.

Their main difference is the point of view; one view it as an opportunity and the other one perceives it as a thread -under some occasion- for the stakeholders.

Continuing on a very different point of view for the mobile ticketing industry, we leave behind the technological aspect of such a solution/implementation and we move into the human/commuter perspective.

Di Pietro et al. (2015) have published a study which examines the users' acceptance and usage for mobile payments, focusing on the mobile ticketing technologies applied in a public transport context

## TECHNOLOGY USED

The Development of the C Language Dennis M. Ritchie Bell Labs/Lucent Technologies Murray Hill, NJ 07974 USA dmr@bell-labs.com  
ABSTRACT The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system

Derived from the typeless language BCPL, it evolved a type structure; created on a tiny machine as a tool to improve a meager programming environment, it has become one of the dominant languages of today.

This paper studies its evolution. Introduction This paper is about the development of the C programming language, the influences on it, and the conditions under which it was created. For the sake of brevity, I omit full descriptions of C itself, its parent B [Johnson 73] and its grandparent BCPL [Richards 79], and instead concentrate on characteristic elements of each language and how they evolved.

C came into being in the years 1969-1973, in parallel with the early development of the Unix operating system; the most creative period occurred during 1972.

Another spate of changes peaked between 1977 and 1979, when portability of the Unix system was being demonstrated.

In the middle of this second period, the first widely available description of the language appeared: The C Programming Language, often called the 'white book' or 'K&R' [Kernighan 78].

Finally, in the middle 1980s, the language was officially standardized by the ANSI X3J11 committee, which made further changes. Until the early 1980s, although compilers existed for a variety of machine architectures and operating systems, the language was almost exclusively associated with Unix; more recently, its use has spread much more widely, and today it is among the languages most commonly used throughout the computer industry. History:

the setting The late 1960s were a turbulent era for computer systems research at Bell Telephone Laboratories [Ritchie 78] [Ritchie 84]. The company was pulling out of the Multics project [Organick 75], which had started as a joint venture of MIT, General Electric, and Bell Labs; by 1969, Bell Labs management, and even the researchers, came to believe that the promises of Multics could be fulfilled only too late and too expensively.

Even before the GE-645 Multics machine was removed from the premises, an informal group, led primarily by Ken Thompson, had begun investigating alternatives.

Thompson wanted to create a comfortable computing environment constructed according to his own design, using whatever means were available.

His plans, it is evident in retrospect, \_\_\_\_\_ Copyright 1993 Association for Computing Machinery, Inc. This electronic reprint made available by the author as a courtesy. For further publication rights contact ACM or the author.

This article was presented at Second History of Programming Languages conference, Cambridge, Mass., April, 1993.

Ritchie Development of C 2 incorporated many of the innovative aspects of Multics, including an explicit notion of a process as a locus of control, a tree-structured file system, a command interpreter as user-level program, simple representation of text files, and generalized access to devices.

They excluded others, such as unified access to memory and to files.

At the start, moreover, he and the rest of us deferred another pioneering (though not original) element of Multics, namely writing almost exclusively in a higher-level language. PL/I, the implementation language of Multics, was not much to our tastes, but we were also using other languages, including BCPL, and we regretted losing the advantages of writing programs in a language above the level of assembler, such as ease of writing and clarity of understanding.

At the time we did not put much weight on portability; interest in this arose later. Thompson was faced with a hardware environment cramped and spartan even for the time: the DEC PDP-7 on which he started in 1968 was a machine with 8K 18-bit words of memory and no software useful to him. While wanting to use a higher-level language, he wrote the original Unix system in PDP-7 assembler. At the start, he did not even program on the PDP-7 itself, but instead used a set of macros for the GEMAP assembler on a GE-635 machine.

A postprocessor generated a paper tape readable by the PDP-7. These tapes were carried from the GE machine to the PDP-7 for testing until a primitive Unix kernel, an editor, an assembler, a simple shell (command interpreter), and a few utilities (like the Unix `rm`, `cat`, `cp` commands) were completed.

After this point, the operating system was self-supporting: programs could be written and tested without resort to paper tape, and development continued on the PDP-7 itself. Thompson's PDP-7 assembler outdid even DEC's in simplicity; it evaluated expressions and emitted the corresponding bits. There were no libraries, no loader or link editor: the entire source of a program was presented to the assembler, and the output file—`a.out`—with a fixed name—that emerged was directly executable. (This name, `a.out`, explains a bit of Unix etymology; it is the output of the assembler.

Even after the system gained a linker and a means of specifying another name explicitly, it was retained as the default executable result of a compilation.) Not long after Unix first ran on the PDP-7, in 1969, Doug McIlroy created the new system's first higher-level language: an implementation of McClure's TMG [McClure 65].

TMG is a language for writing compilers (more generally, TransMoGrifiers) in a top-down, recursive-descent style that combines context-free syntax notation with procedural elements. McIlroy and Bob Morris had used TMG to write the early PL/I compiler for Multics. Challenged by McIlroy's feat in reproducing TMG, Thompson decided that Unix—possibly it had not even been named yet—needed a system programming language. After a rapidly scuttled attempt at Fortran, he created instead a language of his own, which he called B.

B can be thought of as C without types; more accurately, it is BCPL squeezed into 8K bytes of memory and filtered through Thompson's brain. Its name most probably represents a contraction of BCPL, though an alternate theory holds that it derives from Bon [Thompson 69], an unrelated language created by Thompson during the Multics days.

Bon in turn was named either after his wife Bonnie, or (according to an encyclopedia quotation in its manual), after a religion whose rituals involve the murmuring of magic formulas.

Origins: the language BCPL was designed by Martin Richards in the mid-1960s while he was visiting MIT, and was used during the early 1970s for several interesting projects, among them the OS6 operating system at Oxford [Stoy 72], and parts of the seminal Alto work at Xerox PARC [Thacker 79].

We became familiar with it because the MIT CTSS system [Corbato 62] on which Richards worked was used for Multics development. The original BCPL compiler was transported both to Multics and to the GE-635 GECOS system by Rudd Canaday and others at Bell Labs [Canaday 69]; during the final throes of Multics's life at Bell Labs and immediately after, it was the language of choice among the group of people who would later become involved with Unix. BCPL, B, and C all fit firmly in the traditional procedural family typified by Fortran and Ritchie Development of C 3 Algol 60.

They are particularly oriented towards system programming, are small and compactly described, and are amenable to translation by simple compilers.

They are 'close to the machine' in that the abstractions they introduce are readily grounded in the concrete data types and operations supplied by conventional computers, and they rely on library routines for input-output and other interactions with an operating system.

With less success, they also use library procedures to specify interesting control constructs such as coroutines and procedure closures. At the same time, their abstractions lie at a sufficiently high level that, with care, portability between machines can be achieved. BCPL, B and C differ syntactically in many details, but broadly they are similar. Programs consist of a sequence of global declarations and function (procedure) declarations.

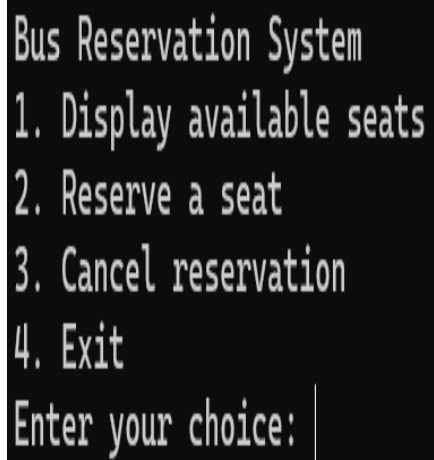
Procedures can be nested in BCPL, but may not refer to non-static objects defined in containing procedures. B and C avoid this restriction by imposing a more severe one: no nested procedures at all.

Each of the languages (except for earliest versions of B) recognizes separate compilation, and provides a means for including text from named files. Several syntactic and lexical mechanisms of BCPL are more elegant and regular than those of B and C. For example, BCPL's procedure and data declarations have a more uniform structure, and it supplies a more complete set of looping constructs. Although BCPL programs are notionally supplied from an undelimited stream of characters, clever rules allow most semicolons to be elided after statements that end on a line boundary. B and C omit this convenience, and end most statements with semicolons.

In spite of the differences, most of the statements and operators of BCPL map directly into corresponding B and C. Some of the structural differences between BCPL and B stemmed from limitations on intermediate memory.

For example, BCPL declarations may take the form let P1 be command and P2 be command and P3 be command ... where the program text represented by the commands contains whole procedures .

## OUTPUT SCREEN SHOTS

A screenshot of a terminal window with a dark background. The text is displayed in a light green or cyan monospaced font. It shows a menu for a 'Bus Reservation System' with four numbered options: '1. Display available seats', '2. Reserve a seat', '3. Cancel reservation', and '4. Exit'. Below the menu, the prompt 'Enter your choice: ' is followed by a vertical cursor bar.

```
Bus Reservation System
1. Display available seats
2. Reserve a seat
3. Cancel reservation
4. Exit
Enter your choice: |
```

```
Bus Reservation System
1. Display available seats
2. Reserve a seat
3. Cancel reservation
4. Exit
Enter your choice: 1
```

```
Available seats:
Seat 1: Available
Seat 2: Available
Seat 3: Available
Seat 4: Available
Seat 5: Available
Seat 6: Available
Seat 7: Available
Seat 8: Available
Seat 9: Available
Seat 10: Available
```

```
Bus Reservation System
1. Display available seats
2. Reserve a seat
3. Cancel reservation
4. Exit
Enter your choice: |
```

```
2. Reserve a seat
3. Cancel reservation
4. Exit
Enter your choice: 1
```

```
Available seats:
Seat 1: Available
Seat 2: Available
Seat 3: Available
Seat 4: Available
Seat 5: Available
Seat 6: Available
Seat 7: Available
Seat 8: Available
Seat 9: Available
Seat 10: Available
```

```
Bus Reservation System
1. Display available seats
2. Reserve a seat
3. Cancel reservation
4. Exit
Enter your choice: 2
Enter the seat number you want to reserve (1-10): 6
Seat 6 reserved successfully.
```

```
Bus Reservation System
1. Display available seats
2. Reserve a seat
3. Cancel reservation
4. Exit
Enter your choice: |
```







