باب-4

مسئلہ کے حل کا تعارف
(INTRODUCTION TO PROBLEM SOLVING)

5196CH04

کمپیوٹر سائنس تجرید کا علم ہے یہ کسی مسئلے کا مناسب ماڈل کی تخلیق کرتا ہے اور اسے حل کرنے کی کارگر تکنیکوں کو بھی پیش کرتا ہے۔

-اے. آہو اور جے. المان

(A. Aho and J. Ullman)

اس باب میں

- تعارف
- حل مسائل کے اقدامات
- الگورتھم
- الگورتھم کی نمائندگی
- کنٹرول کا بہاؤ
- الگورتھم کی تصدیق کرنا
- الگورتھم کا موازنہ
- کوڈنگ
- تحلیل

GIGO (کچرا اندر کچرا باہر)

کمپیوٹر کے ذریعے پیش کیے جانے والے آؤٹ پُٹ (نتیجہ) کی درستی کا دارومدار کمپیوٹر کو فراہم کیے جانے والے ان پُٹ کی درستی پر ہوتا ہے۔

4.1 تعارف

آج، کمپیوٹر ہمارے چاروں طرف موجود ہیں۔ ہم ان کا استعمال مختلف کاموں کو تیز رفتار اور بہت زیادہ درستگی کے ساتھ انجام دینے کے لیے کرتے ہیں۔ مثال کے طور پر ہم کمپیوٹر یا اسمارٹ فون کا استعمال کر کے آن لائن ریل گاڑی کے ٹکٹ بُک کرتے ہیں۔

ہندوستان ایک وسیع و عریض ملک ہے اور اس کا ریلوے نیٹ ورک بہت بڑا ہے۔ لہٰذا ریلوے ریزرویشن ایک پیچیدہ عمل ہے۔ ریزرویشن کرنے کے عمل میں کئی پہلوؤں کی معلومات شامل ہے مثلاً ٹرین کی تفصیلات (ٹرین کی قسم، ہر ایک ٹرین میں کمپارٹمنٹ اور برتھ کی قسم، ان کا نظام الاوقات وغیرہ)، متعدد صارفین کے ذریعے بہ یک وقت ٹکٹوں کی بکنگ اور دیگر متعلقہ عوامل۔

کمپیوٹروں کے استعمال کی وجہ سے آج ٹرین ٹکٹوں کی بکنگ آسان ہوگئی ہے۔ ٹرین ٹکٹوں کی آن لائن بکنگ نے ہمیں کسی بھی وقت اور کہیں سے بھی ٹکٹ بک کرانے کی سہولت فراہم کر کے ہماری زندگی کو مزید آسان بنا دیا ہے۔

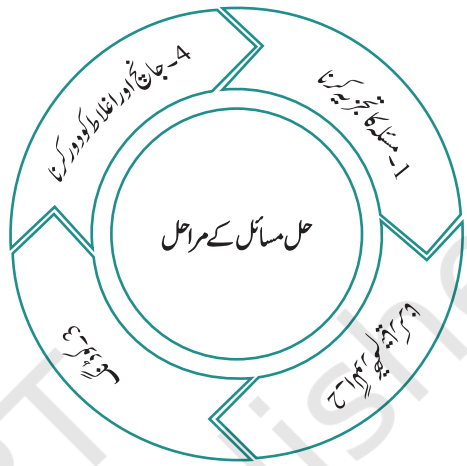
کمپیوٹرائزیشن (Computerisation) اصطلاح کا استعمال ہم عام طور سے کسی بھی انسانی کام کو کارگر طریقے سے خودکار انداز میں انجام دینے کے لیے سافٹ ویئر تیار کرنے کے مقصد سے کمپیوٹر کے استعمال کو ظاہر کرنے کے لیے کرتے ہیں۔ کمپیوٹروں کا استعمال روزمرہ کے مختلف مسائل کو حل کرنے کے لیے کیا جاتا ہے، چنانچہ حل مسائل (Problem Solving) ایک ایسا لازمی ہنر ہے جس سے کمپیوٹر سائنس کے طالب علم کو واقف ہونا چاہیے۔ یہاں یہ بات بھی قابل ذکر ہے کہ کمپیوٹر خود کسی مسئلے کو حل نہیں کر سکتے ہیں۔ کسی مسئلہ کو حل کرنے کے لیے ہمیں اسے مرحلہ وار ہدایات دینی ہوں گی۔ لہٰذا کسی مسئلے کو حل کرنے کے معاملے میں کمپیوٹر کی کامیابی اس بات پر منحصر ہے کہ ہم مسئلے کی وضاحت کتنے صحیح طریقے سے اور درستگی کے ساتھ کرتے ہیں، حل کا خاکہ (الگورتھم) تیار کرتے ہیں اور پروگرامنگ لینگویج کا استعمال کر کے حل (پروگرام) کا نفاذ کرتے ہیں۔ چنانچہ، حل مسائل کسی مسئلے کی شناخت کرنے، اس مسئلے کے لیے الگورتھم تیار کرنے اور آخر میں کمپیوٹر پروگرام کو فروغ دینے کے لیے الگورتھم کو نافذ کرنے کا عمل ہے۔

4.2 حل مسائل کے اقدامات (STEPS FOR PROBLEM SOLVING)

فرض کیجیے کہ گاڑی چلاتے وقت اس میں ایک عجیب آواز پیدا ہونے لگتی ہے۔ ہمیں شاید فوری طور پر یہ معلوم نہ

ہو سکے کہ مسئلے کو کس طرح حل کیا جانا ہے۔ سب سے پہلے ہم یہ جاننے کی کوشش کرتے ہیں کہ آواز کہاں سے آ رہی ہے؟ اگر ہم مسئلے کو حل کرنے سے قاصر رہتے ہیں تو ہم گاڑی کو میکینک کے پاس لے جاتے ہیں۔ میکینک شور کے ماخذ کی شناخت کرنے کے لیے مسئلے کا تجزیہ کرے گا، کیے جانے والے کام کا منصوبہ تیار کرے گا اور آخر میں شور کو ختم کرنے کے لیے گاڑی کی مرمت کرے گا۔ مذکورہ بالا مثال سے یہ بات واضح ہے کہ کسی مسئلے کا حل تلاش کرنے کے عمل میں کئی مراحل شامل ہو سکتے ہیں۔

جب مسائل آسان اور سادہ نوعیت کے ہوں تو ہم ان کا حل آسانی سے تلاش کر سکتے ہیں۔ لیکن ایک پیچیدہ مسئلے کا درست حل تلاش کرنے کے لیے ایک باضابطہ اور منظّم طریق کار اختیار کرنے کی ضرورت ہے ۔ بالفاظ دیگر، ہمیں حل مسائل کی تکنیکوں کو نافذ کرنا ہوگا۔ مسئلے کو حل کرنے کا عمل مسئلے کی درست شناخت کے ساتھ شروع ہوتا ہے اور ایک پروگرام یا سافٹ ویئر کے ضمن میں مسئلے کے مکمل اور کارگر حل کے ساتھ ختم ہوتا ہے۔ کمپیوٹر کا استعمال کر کے کسی مسئلے کو حل کرنے کے لیے درکار کلیدی مراحل شکل 4.1 میں دکھائے گئے ہیں اور ان پر ذیل کے سیکشنوں میں بحث کی گئی ہے۔



شکل 1.4: مسئلے کو حل کرنے کے مراحل

4.2.1 مسئلے کا تجزیہ کرنا (Analysing the Problem)

کسی مسئلے کا حل تلاش کرنے سے پہلے اسے صریح طور پر سمجھنا بہت اہم ہے۔ اگر ہمیں یہ بات واضح طور پر معلوم نہیں ہے کہ کیا حل کرنا ہے، تو ہم ایسا پروگرام تیار کر لیں گے کہ جس سے ہمارا مقصد حل نہیں ہوگا۔ لہٰذا، ہمیں مسئلے کے اہم اجزا کی فہرست بناتے وقت مسئلے کے بیان کو غور سے پڑھنے اور اس کا تجزیہ کرنے کی ضرورت ہے اور اس بات کا تعین کرنا ہے کہ ہمارے حل میں بنیادی نوعیت کے کون سے کام شامل ہونے چاہئیں۔ کسی مسئلے کا تجزیہ کر کے ہم یہ معلوم کرنے کے اہل ہوں گے کہ وہ کون سے ان پُٹ ہیں جنھیں ہمارے پروگرام کو حاصل کرنا چاہیے اور وہ کون سے آؤٹ پُٹ ہیں جو اسے پیش کرنے چاہئیں۔

4.2.2 الگورتھم کو فروغ دینا (Developing an Algorithm)

کسی دیے ہوئے مسئلے کے لیے پروگرام یا کوڈ لکھنے سے پہلے حل کی تدبیر لازمی ہے۔ حل کو فطری زبان میں ظاہر کیا جاتا ہے اور اسے الگورتھم کہتے ہیں۔ ہم الگورتھم کو ڈش تیار کرنے کی ایسی ترکیب تصور کر سکتے ہیں جسے عمدہ طریقے سے تحریر کیا گیا ہو اور اس کے مراحل واضح طور پر متعین ہوں کہ اگر کوئی ان پر عمل کرے تو مذکورہ ڈش تیار کر سکتا ہے۔

ہم اس حل کی ابتدا ایک عارضی یا مجوزہ منصوبے سے کرتے ہیں اور الگورتھم کو اس وقت تک نکھارتے رہتے ہیں جب تک کہ الگورتھم مطلوبہ حل کے سبھی پہلوؤں پر گرفت نہ حاصل کر لے۔ کسی دیے ہوئے مسئلے کے لیے ایک سے زیادہ الگورتھم ممکن ہیں اور ہمیں مناسب ترین حل کا انتخاب کرنا ہوگا۔ سیکشن 4.3 میں الگورتھم سے بحث کی گئی ہے۔

الگورتھم

الگورتھم بالکل درست مراحل کا ایسا مجموعہ ہے جس پر عمل کرنے کے نتیجے میں مسئلے کا حل نکلتا ہے یا مطلوبہ کام انجام کو پہنچتا ہے۔

4.2.3 کوڈنگ (Coding)

الگورتھم کو حتمی شکل دینے کے بعد ہمیں الگورتھم کو ایک ایسی شکل میں تبدیل کرنے کی ضرورت ہے جسے کمپیوٹر سمجھ سکے اور مطلوبہ حل فراہم کر سکے۔ پروگرام لکھنے کے لیے مختلف اعلیٰ سطحی پروگرامنگ لینگویج کا استعمال کیا جاسکتا ہے۔

کوڈنگ کے طریقہ کار کی تفصیلات کو ریکارڈ کرنا اور حل کی دستاویز سازی بھی اتنی ہی اہمیت کی حامل ہے۔ یہ بعد کے مرحلے میں پروگرام پر نظر ثانی کے دوران بہت مفید ثابت ہوتا ہے۔ کوڈنگ کو سیکشن 4.8 میں مفصل بیان کیا گیا ہے۔

4.2.4 جانچ اور اغلاط کو دور کرنا (Testing and Debugging)

مختلف معیارات کی بنیاد پر تیار کیے گئے پروگرام کی جانچ کی جانی چاہیے۔ پروگرام ایسا ہونا چاہیے کہ وہ استعمال کنندہ کی ضروریات کو پورا کرتا ہو۔ اسے متوقع مدت کے اندر ردعمل کا اظہار کرنا چاہیے۔ اسے سبھی ممکنہ ان پُٹ کے لیے صحیح آؤٹ پُٹ پیش کرنا چاہیے۔ نحوی اغلاط کی موجودگی میں کوئی آؤٹ پُٹ حاصل نہیں ہوگا۔ اگر تشکیل شدہ آؤٹ پُٹ غلط ہے تو پروگرام کی جانچ کر کے منطقی اغلاط (اگر کوئی ہے) کو دور کیا جانا چاہیے۔

سافٹ ویئر کی صنعت میں پیچیدہ نوعیت کی اپلی کیشن کو تیار کرنے کے دوران جانچ کے معیاری طریقے بروئے کار لائے جاتے ہیں مثلاً اکائی یا جزو ترکیبی جانچ (Unit or Component Testing)، مربوط جانچ (Integration Testing)، سسٹم جانچ (System Testing) اور قبولیت جانچ (Acceptance Testing)۔ یہ اس بات کو یقینی بنانے کے لیے ہے کہ سافٹ ویئر سبھی کاروباری اور تکنیکی ضروریات کو پورا کرتا ہے اور توقع کے مطابق کام کرتا ہے۔ جانچ کے مرحلے میں پائی گئی اغلاط یا نقائص کو دور کیا جاتا ہے اور پروگرام کی دوبارہ جانچ کی جاتی ہے۔ یہ عمل اس وقت تک جاری رہتا ہے جب تک کہ پروگرام کی سبھی غلطیاں دور نہیں ہو جاتیں۔

سافٹ ویئر اپلی کیشن کے تیار ہو جانے، اس کی جانچ ہو جانے اور استعمال کنندہ کے حوالے ہو جانے کے بعد بھی اس کے کام کرنے کے دوران کچھ مسائل پیش آسکتے ہیں جنھیں وقتاً فوقتاً حل کرنے کی ضرورت ہوتی ہے۔ سافٹ ویئر کے رکھ رکھاؤ میں استعمال کنندہ کو پیش آنے والی دشواریوں کو دور کرنا، استعمال کنندہ کے سوالوں کے جواب دینا اور خصوصیات میں اضافہ یا ترمیم کی درخواست پر عمل درآمد شامل ہے۔

سرگرمی 4.1

دو اعداد کا LCM معلوم کرنے کے لیے آپ اقدامات کے کون سے سلسلے کا اتباع کریں گے؟

4.3 الگورتھم (ALGORITHM)

ہم اپنی روزمرہ زندگی میں ایسی بہت سی سرگرمیاں انجام دیتے ہیں جن میں ہم مراحل کے ایک مخصوص تسلسل کا اتباع کرتے ہیں۔ اسکول کے لیے تیار ہونا، ناشتہ تیار کرنا، سائیکل چلانا، ٹائی پہننا، معمہ کو حل کرنا وغیرہ اس قسم کی سرگرمیوں کی مثالیں ہیں۔ ہر ایک سرگرمی کو مکمل کرنے کے لیے ہم مراحل کے ایک تسلسل کا اتباع کرتے ہیں۔

فرض کیجیے کہ مندرجہ ذیل مراحل سائیکل چلانے کے لیے درکار ہیں۔

1) سائیکل کو اسٹینڈ سے الگ کریں
2) سائیکل کی سیٹ پر بیٹھیں
3) پیڈل گھمائیں
4) حسب ضرورت بریک لگائیں
5) منزل پر پہنچ کر رکیں

آیئے اب دو اعداد 45 اور 46 کا عاد اعظم مشترک (HCF) معلوم کریں۔

نوٹ: عاد اعظم وہ سب سے بڑا عدد ہے جو دیے ہوئے دونوں اعداد کو تقسیم کرتا ہے۔

مرحلہ 1: وہ اعداد (قاسم) معلوم کیجیے جو دیے ہوئے اعداد کو تقسیم کر سکتے ہیں۔

45 کے قاسم ہیں: 1، 3، 5، 9، 15 اور 45

54 کے قاسم ہیں: 1، 2، 3، 6، 9، 18، 27 اور 54

مرحلہ 2: اب ان دونوں فہرستوں میں سے سب سے بڑا مشترک عدد معلوم کیجیے۔

لہٰذا، 45 اور 46 کا عاد اعظم مشترک 9 ہے۔

چنانچہ یہ بات واضح ہے کہ کام کو مکمل کرنے کے لیے ہمیں مراحل کے تسلسل کا اتباع کرنے کی ضرورت ہے۔ مطلوبہ نتائج (Output) حاصل کرنے کے لیے درکار مراحل کے ایسے متناہی تسلسل کو الگورتھم کہتے ہیں۔ اگر صحیح طریقے سے عمل کیا جائے تو یہ ایک متناہی مدت میں مطلوبہ نتیجہ فراہم کرے گا۔ الگورتھم کی ایک متعین ابتدا اور انتہا ہوتی ہے اور اس کے مراحل کی تعداد بھی متناہی ہوتی ہے۔

اصطلاح 'الگورتھم' کی ابتدا فارس کے ماہر فلکیات اور ریاضی داں ابوعبداللہ محمد ابنِ موسیٰ الخوارزمی (c. 850 AD) سے ہوئی ہے کیوں کہ الخوارزمی کے لاطینی ترجمہ کو 'الگورتھمی' کہا جاتا تھا۔

4.3.1 ہمیں الگورتھم کی ضرورت کیوں ہے؟

ایک پروگرامر کمپیوٹر کو ہدایات دینے کے لیے پروگرام تحریر کرتا ہے تاکہ کسی مخصوص کام کو حسب منشا انجام دیا جا سکے۔ اس کے بعد کمپیوٹر پروگرام کوڈ میں لکھے ہوئے مراحل کا اتباع کرتا ہے۔ لہٰذا، ایک پروگرامر کوڈ کو حقیقی شکل دینے سے پہلے تحریر کیے جانے والے پروگرام کا خاکہ تیار کرتا ہے۔ خاکہ تیار کیے بغیر کوئی بھی پروگرامر اپنے ذہن میں تحریر کی جانے والی ہدایات کا تصور پیدا نہیں کر سکتا ہے جس کے نتیجے میں ایک ایسا پروگرام تیار ہو جائے گا جو توقع کے مطابق کارگر ثابت نہ ہو۔

اس قسم کا خاکہ کچھ اور نہیں بلکہ ایک الگورتھم ہے جو کمپیوٹر پروگرام کا بلڈنگ بلاک ہے۔ مثال کے طور پر سرچ انجن کا استعمال کر کے تلاش کرنا، پیغام ارسال کرنا، دستاویز میں کوئی لفظ تلاش کرنا، ایپ کی مدد سے ٹیکسی بک کرنا، آن لائن بینکنگ کرنا، کمپیوٹر گیم کھیلنا، یہ سبھی الگورتھم پر مبنی ہیں۔

الگورتھم تحریر کرنے کا عمل اکثر و بیشتر پروگرام کا پہلا مرحلہ تصور کیا جاتا ہے۔ جب کسی مسئلے کو حل کرنے کے لیے ہمارے پاس الگورتھم موجود ہو تو ہم کمپیوٹر کو ہدایات دینے کے لیے اعلیٰ سطحی زبان (HLL) میں کمپیوٹر

نوٹ

پروگرام تحریر کر سکتے ہیں۔ اگر الگورتھم درست ہے تو کمپیوٹر ہر مرتبہ پروگرام کو صحیح طریقے سے انجام دے گا۔ چنانچہ الگورتھم کو استعمال کرنے کا مقصد کسی مسئلے کا حل تلاش کرنے کے عمل کی معتربیت، درستگی اور کارکردگی میں اضافہ کرنا ہے۔

(A) اچھے الگورتھم کی خصوصیات

- درستگی — مراحل کو درستگی کے ساتھ بیان یا متعین کیا گیا ہو۔
- منفرد نوعیت (Uniqueness) — ہر ایک مرحلے کے نتائج منفرد انداز میں متعین ہوتے ہیں اور ان کا انحصار صرف ان پُٹ اور سابقہ مرحلوں کے نتیجے پر ہوتا ہے۔
- متناہیت (Finiteness) — الگورتھم ہمیشہ مراحل کی ایک متناہی تعداد کے بعد رک جاتا ہے۔
- ان پُٹ (Input) — الگورتھم ان پُٹ حاصل کرتا ہے۔
- آؤٹ پُٹ (Output) — الگورتھم آؤٹ پُٹ فراہم کرتا ہے۔

(B) الگورتھم تحریر کرتے وقت مندجہ ذیل کی واضح نشاندہی ضروری ہے:

- استعمال کنندہ سے حاصل کیا جانے والا ان پُٹ
- مطلوبہ نتیجہ حاصل کرنے کے لیے انجام دیا جانے والا پروسیسنگ یا کمپیوٹیشن کا عمل
- استعمال کنندہ کو مطلوب آؤٹ پُٹ

4.4 الگورتھم کی نمائندگی (Representation of Algorithms)

سافٹ ویئر تیار کرنے والے افراد یا پروگرامر الگورتھم سے متعلق اپنی فکری مہارتوں کا استعمال کر کے مسئلے کا تجزیہ کرتے ہیں اور ان منطقی مراحل کی شناخت کرتے ہیں جنھیں بروئے کار لا کر مسئلے کے حل تک پہنچا جا سکتا ہے۔ ایک مرتبہ مراحل کی شناخت ہو جانے کے بعد ان مراحل کو درکار ان پُٹ اور مطلوب آؤٹ پُٹ کے ساتھ لکھا جاتا ہے۔ الگورتھم کے اظہار کے دو عام طریقے ہیں- فلو چارٹ اور سوڈو کوڈ۔ الگورتھم کے اظہار کے لیے مندرجہ ذیل کو ذہن میں رکھتے ہوئے ان میں سے کسی بھی طریقے کا استعمال کیا جا سکتا ہے۔

- یہ نفاذ سے متعلق کسی بھی قسم کی تفصیل کے علاوہ مسئلے کے حل کی منطق کو ظاہر کرتا ہے۔
- یہ پروگرام پر عمل درآمد کے دوران کنٹرول کے بہاؤ کو واضح طور پر ظاہر کرتا ہے۔

4.4.1 فلو چارٹ — الگورتھم کا بصری اظہار

فلو چارٹ دراصل الگورتھم کا بصری اظہار ہے۔ فلو چارٹ ایک ایسا ڈائیگرام ہے جو تیر کے نشان کے ذریعے باہم منسلک باکس، ڈائمنڈ اور دیگر اشکال سے بنا ہوتا ہے۔ ہر شکل حل سے متعلق علم ایک مرحلے کی نمائندگی کرتی ہے اور تیر کا نشان ان مراحل کی ترتیب یا ان کے درمیان ربط کو ظاہر کرتا ہے۔ فلو چارٹ بنانے کے لیے معیاری علامتیں استعمال کی جاتی ہیں۔ ان میں سے کچھ جدول 4.1 میں دی ہوئی ہیں۔

جدول 4.1 فلوچارٹ بنانے کے لیے اشکال یا علامتیں

فلوچارٹ کی علامات	فنکشن	وضاحت
	ابتدا/اختتام	اسے ''ٹرمینیٹر'' علامت بھی کہا جاتا ہے۔ یہ اس بات کی طرف اشارہ کرتا ہے کہ فلوچارٹ کی ابتدا کہاں سے ہوتی ہے اور یہ کہاں پر اختتام پذیر ہوتا ہے۔
	عمل	اسے ''ایکشن'' علامت بھی کہا جاتا ہے۔ یہ کسی عمل، کارروائی یا واحد مرحلے کی نمائندگی کرتا ہے۔
	فیصلہ	فیصلہ یا شاخدار نقطہ، عام طور سے ہاں/نہیں یا صحیح/غلط سے متعلق سوال پوچھا جاتا ہے اور جواب کی بنیاد پر راستہ دو شاخوں میں تقسیم ہو جاتا ہے۔
	ان پُٹ/آؤٹ پُٹ	اسے ڈیٹا کی علامت بھی کہا جاتا ہے۔ اس متوازی الاضلاع کا استعمال ڈیٹا کے ان پُٹ یا آؤٹ پُٹ کے لیے کیا جاتا ہے۔
	تیر	اشکال کے درمیان بہاؤ (Flow) کی ترتیب کو ظاہر کرنے کے لیے کنکٹر

مثال 4.1 کسی عدد کا مربع معلوم کرنے کے لیے الگورتھم تحریر کیجیے۔

الگورتھم تحریر کرنے سے پہلے آیئے ان پُٹ، عمل (پروسیس) اور آؤٹ پُٹ کی نشان دہی کریں:

- ان پُٹ: وہ عدد جس کا مربع معلوم کیا جانا ہے
- عمل (پروسیس): مربع حاصل کرنے کے لیے عدد کو خود سے ضرب کرنا
- آؤٹ پُٹ: عدد کا مربع

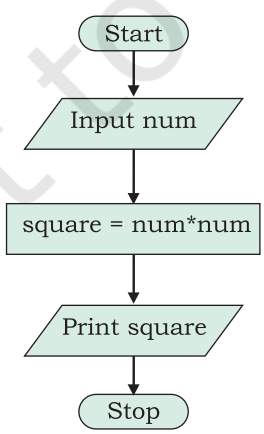
عدد کا مربع معلوم کرنے کے لیے الگورتھم

پہلا مرحلہ: ایک عدد ان پُٹ کیجیے اور اسے num میں اسٹور کیجیے

دوسرا مرحلہ: num * num کی تحسیب کیجیے اور اسے square میں اسٹور کیجیے

تیسرا مرحلہ: square کو پرنٹ کیجیے

کسی عدد کا مربع معلوم کرنے کے لیے الگورتھم کو فلوچارٹ کا استعمال کر کے تصویر کی شکل میں ظاہر کیا جا سکتا ہے۔ جیسا کہ شکل 4.2 میں دکھایا گیا ہے۔



شکل 4.2 عدد کا مربع معلوم کرنے کے لیے فلوچارٹ

سوچیے اور جواب دیجیے

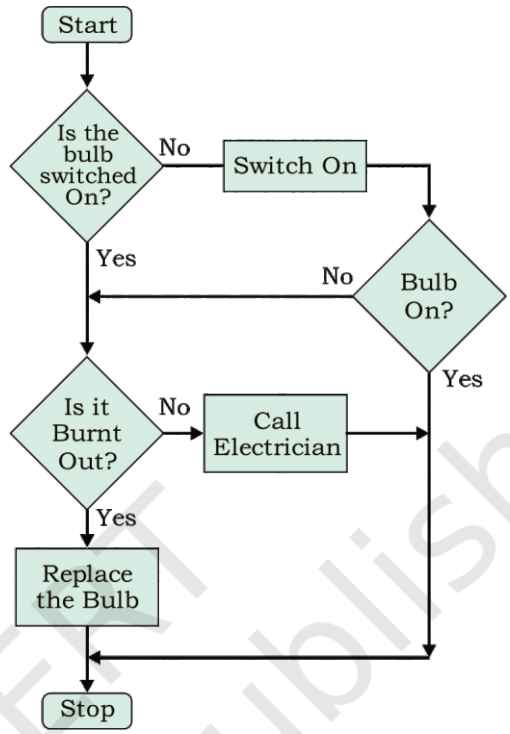
اگر کوئی الگورتھم ایک متناہی تعداد والے مراحل کے بعد نہیں رکتا ہے تو کیا ہوگا؟

سرگرمی 4.2

ایک فلوچارٹ بنایئے جو آپ کے کیریئر سے متعلق اہداف کے حصول کی نمائندگی کرتا ہے۔

نوٹ

مثال 4.2 روشنی کا ایک بلب کام نہیں کر رہا ہے۔اس مسئلے کو حل کرنے کے لیے فلوچارٹ بنائیے۔



شکل 4.3 : روشنی کے بلب کے ناکارہ ہوجانے کے مسئلے کو حل کرنے کے لیے فلوچارٹ

4.4.2 سوڈوکوڈ (Pseudocode)

سوڈوکوڈ الگورتھم کو ظاہر کرنے کا ایک اور طریقہ ہے۔اسے ایک ایسی غیر رسمی زبان تصور کیا جاتا ہے جو الگورتھم تحریر کرنے میں پروگرامر کی مدد کرتی ہے۔یہ ہدایات کا ایسا تفصیلی بیان ہے جس پر کمپیوٹر کو ایک خاص ترتیب میں عمل کرنا چاہیے۔اسے انسان پڑھ سکتے ہیں لیکن براہ راست کمپیوٹر کے ذریعے ایگزیکیوٹ نہیں کیا جاسکتا ہے۔سوڈو کوڈ لکھنے کے لیے کوئی مخصوص معیار موجود نہیں ہے۔لفظ ''Pseudo'' کا مطلب ہے ''Not Real'' یعنی غیر حقیقی۔ چنانچہ ''Pseudocode'' کا مطلب ہے ایسا کوڈ جو حقیقی کوڈ نہیں ہے۔سوڈوکوڈ لکھتے وقت بار بار استعمال ہونے والے کچھ کلیدی الفاظ ذیل میں دیے گئے ہیں۔

- INPUT
- COMPUTE
- PRINT
- INCREMENT
- DECREMENT
- IF/ELSE
- WHILE
- TRUE/FALSE

سرگرمی 4.3

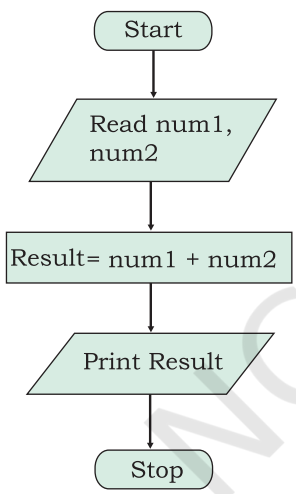
ہاکی میچ کے اسکور بورڈ کی تشکیل کے لیے ایک سوڈوکوڈ لکھیے۔

مثال 4.3 سوڈوکوڈ اور فلوچارٹ دونوں کا استعمال کر کے صارف کے ذریعے داخل کیے گئے دو اعداد کے حاصل جمع کو ظاہر کرنے کے لیے ایک الگورتھم لکھیے۔

دو اعداد کے حاصل جمع کے لیے سوڈوکوڈ مندرجہ ذیل ہوگا:

```
INPUT num1
INPUT num2
COMPUTE Result = num1 + num2
PRINT Result
```

اس الگورتھم کے لیے فلوچارٹ شکل 4.4 میں دیا گیا ہے۔



شکل 4.4 : دو اعداد کے حاصل جمع کو ظاہر کرنے کے لیے فلوچارٹ

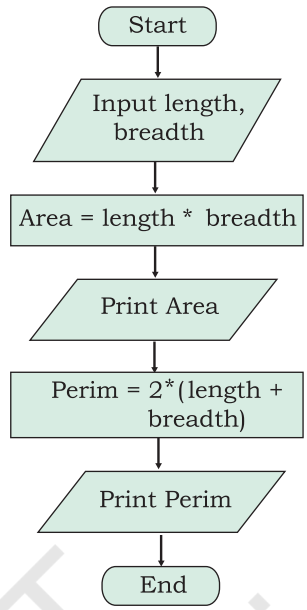
مثال 4.4 سوڈوکوڈ اور فلوچارٹ دونوں کا استعمال کرتے ہوئے مستطیل کے رقبے اور احاطے کی تحسیب کے لیے ایک الگورتھم لکھیے۔

مستطیل کے رقبے اور احاطے کی تحسیب کے لیے سوڈوکوڈ

```
INPUT length
INPUT breadth
COMPUTE Area = length * breadth
PRINT Area
COMPUTE Perim = 2 * (length + breadth)
PRINT Perim
```

اس الگورتھم کے لیے فلوچارٹ شکل 4.5 میں دیا گیا ہے۔

نوٹ



شکل 4.5 : مستطیل کے رقبے اور احاطے کی تحسیب کے لیے فلوچارٹ

(A) سوڈو کوڈ کے فوائد (Benefits of Pseudocode)

اعلیٰ سطحی زبان میں کوڈ لکھنے سے پہلے، پروگرام کا سوڈو کوڈ مطلوبہ پروگرام کی بنیادی کارکردگی (Functionality) کو ظاہر کرنے میں مدد کرتا ہے۔ کوڈ کو پہلے انسانوں کے ذریعے پڑھی جاسکنے والی زبان میں لکھنے سے پروگرامر کو کسی بھی اہم مرحلے کے چھوٹ جانے کا اندیشہ نہیں رہتا۔ علاوہ ازیں، جو لوگ پروگرامر نہیں ہیں ان کے لیے حقیقی پروگرام کو پڑھنا اور سمجھنا مشکل ہوتا ہے لیکن سوڈو کوڈ کی مدد سے وہ مراحل پر نظر ثانی کر کے اس بات کی تصدیق کر سکتے ہیں کہ مجوزہ عمل مطلوبہ آؤٹ پٹ فراہم کرے گا۔

4.5 کنٹرول کا بہاؤ (Flow of Control)

کنٹرول کا بہاؤ واقعات کے بہاؤ کو ظاہر کرتا ہے جیسا کہ فلوچارٹ میں ظاہر کیا گیا ہے۔ واقعات کا بہاؤ ایک تسلسل میں یا کسی فیصلے کی بنیاد پر مختلف شاخوں میں ہوسکتا ہے یا کچھ حصوں کی متناہی تعداد میں تکرار کرسکتا ہے۔

4.5.1 تسلسل (Sequence)

اگر ہم مثال 4.3 اور 4.4 پر غور کریں تو ہم دیکھیں گے کہ بیانات کو یکے بعد دیگرے یعنی ایک تسلسل میں ایگزیکیوٹ کیا جاتا ہے۔ ایسے الگورتھم جہاں سبھی مراحل یکے بعد دیگرے ایگزیکیوٹ ہوتے ہیں تو یہ کہا جاتا ہے کہ ان کا ایگزیکیوشن تسلسل میں ہے۔ حالاں کہ ایسا نہیں ہے کہ الگورتھم میں سبھی بیانات ہمیشہ ہی تسلسل میں ایگزیکیوٹ ہوں۔ بعض اوقات ہمیں ایسے الگورتھم کی ضرورت ہوتی ہے جو کچھ معمول کے کاموں کو مکرر انداز میں انجام دے یا سابقہ مراحل کے نتائج کی بنیاد پر مختلف طرز عمل کا اظہار کرے۔ اس سیکشن میں ہم یہ سیکھیں گے کہ اس قسم کی صورت حال میں الگورتھم کس طرح تحریر کرنا ہے۔

4.5.2 انتخاب (Selection)

سوچیے اور جواب دیجیے

کیا آپ اپنی روزمرہ زندگی کے معمول سے متعلق ایسی سرگرمیوں کی فہرست بنا سکتے ہیں جن میں فیصلہ سازی کا عمل شامل ہو۔

ایک پڑوس کے نقشے پر غور کریں جیسا کہ شکل 4.6 میں دکھایا گیا ہے۔ فرض کیجیے کہ لال چھت والی گلابی عمارت ایک اسکول ہے، نقشے کے سب سے دور والے سرے پر پیلے رنگ کی عمارت ایک گھر ہے۔

شکل 4.6 : حقیقی زندگی میں فیصلہ سازی

شکل 4.6 کے حوالے سے آیئے مندرجہ ذیل سوالوں کے جواب دیں:

- کیا گھر سے اسکول جانے کے لیے کوئی ایسا راستہ موجود ہے جو پہلے سے متعین ہے؟
- کیا واپس آتے وقت ہم کوئی دوسرا راستہ اختیار کر سکتے ہیں؟

جیسا کہ نقشے میں دیکھا جا سکتا ہے، اسکول اور گھر کے درمیان کئی راستے ہو سکتے ہیں۔ ہم صبح کے وقت سب سے چھوٹا راستہ اختیار کر سکتے ہیں۔ لیکن دوپہر میں گھر واپس آتے وقت سب سے چھوٹے راستے پر بہت زیادہ ٹریفک ہو سکتا ہے۔ چنانچہ ہم کم ٹریفک والا دوسرا راستہ اختیار کر سکتے ہیں۔ لہٰذا مذکورہ بالا مسئلے میں مخصوص شرائط کی بنیاد کچھ فیصلے لینے کی ضرورت ہے۔ آیئے کچھ اور ایسی مثالوں پر غور کریں جہاں فیصلہ سازی کا عمل مخصوص شرائط پر منحصر ہے۔ مثلاً

(i) ووٹ ڈالنے کے لیے اہلیت کی جانچ ***(Checking eligibility for voting)***

عمر کی بنیاد پر کسی فرد کو یا تو ووٹ ڈالنے کی اجازت ہوگی یا نہیں ہوگی:

- اگر عمر 18 سال سے زیادہ یا اس کے مساوی ہے تو وہ فرد ووٹ ڈالنے کے اہل ہے۔
- اگر عمر 18 سال سے کم ہے تو وہ فرد ووٹ ڈالنے کے اہل نہیں ہے۔

(ii) آئیے ایک اور مثال پر غور کریں

اگر ایک طالب علم کی عمر 8 سال ہے اور اسے ریاضی پسند ہے تو
طالب علم کو گروپ A میں رکھیے
بصورت دیگر
طالب علم کو گروپ B میں رکھیے
مذکورہ بالا شرط کے مطابق یہ طلبا کس گروپ میں جائیں گے؟

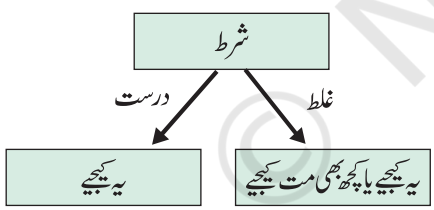
نتیجہ

- 8 سالہ روی جسے ریاضی پسند نہیں ہے: گروپ B
- 8 سالہ پریتی جسے ریاضی پسند ہے: گروپ A
- 7 سالہ انیس جسے ریاضی پسند نہیں ہے: گروپ B

ان مثالوں میں، کسی ایک متبادل کا انتخاب کسی شرط کے نتیجے کی بنیاد پر کیا جاتا ہے۔ امکانات کی جانچ کے لیے شرائط پر مبنی بیانات کا استعمال کیا جاتا ہے۔ پروگرام ایک یا ایک سے زیادہ شرائط کی جانچ کرتا ہے اور شرط کے صحیح (True) یا غلط (False) ہونے کی بنیاد پر عملوں (عملوں کے تسلسل) کو انجام دیتا ہے۔ یہ صحیح (True) یا غلط (False) قدریں بائنری قدریں کہلاتی ہیں۔

الگورتھم میں مشروط بیانات کو مندرجہ ذیل طریقے سے لکھا جا سکتا ہے:

```
If <condition> then
    steps to be taken when the
condition is true/fulfilled
```



شکل 4.7: شرط کے صحیح یا غلط ہونے پر منحصر ہوتے ہیں۔

ایسی بھی صورت حال ہیں جہاں ہمیں شرط پوری نہ ہونے پر بھی کارروائی کرنے کی ضرورت ہوتی ہے (شکل 4.7)۔ اس کی نمائندگی کرنے کے لیے ہم لکھ سکتے ہیں کہ:

```
If <condition> is true then
    steps to be taken when the condition is
true/fulfilled
otherwise
    steps to be taken when the condition is
false/not fulfilled
```

پروگرامنگ کی زبانوں میں انگریزی لفظ Otherwise بمعنی 'بصورت دیگر' کو ظاہر کرنے کے لیے Else کا استعمال کیا جاتا ہے جو ایک کلیدی لفظ (Keyword) ہے۔ چنانچہ، حقیقی پروگراموں میں True/False والے مشروط بیانات کو if-else بلاک کا استعمال کر کے لکھا جاتا ہے۔

مثال 4.5 آیئے اس بات کی جانچ کرنے کے لیے ایک الگورتھم لکھیں کہ کوئی عدد طاق ہے یا جفت۔

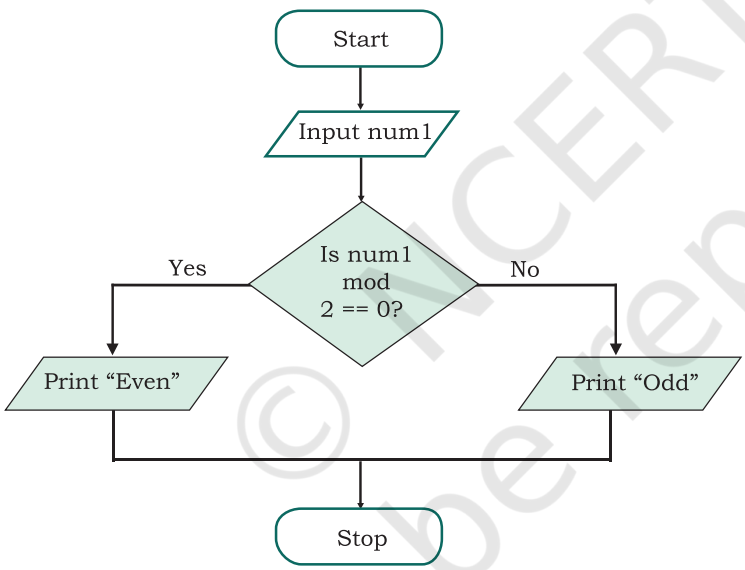
نوٹ

- ان پُٹ: کوئی بھی عدد
- پروسیس (عمل): جانچ کیجیے کہ آیا عدد جفت ہے یا نہیں
- آؤٹ پُٹ: پیغام ''جفت'' یا ''طاق''

الگورتھم کے سوڈوکوڈ کو مندرجہ ذیل طریقے سے لکھا جاسکتا ہے۔

```
PRINT "Enter the Number"
INPUT number
IF number MOD 2 == 0 THEN
     PRINT "Number is Even"
ELSE
     PRINT "Number is Odd"
```

الگورتھم کی نمائندگی کرنے والے فلوچارٹ کو شکل 4.8 میں دکھایا گیا ہے۔



شکل 4.8: کسی عدد کے طاق ہے یا جفت ہونے کی جانچ کرنے کے لیے فلوچارٹ

مثال 4.6 آیئے ہم ایک سوڈوکوڈ لکھیں اور ایک فلوچارٹ بنائیں جہاں معینہ عمر کی بنیاد پر کسی فرد کی بچہ (<13)، نوجوان (13=< لیکن 20>) یا (بالغ 20=<) کے طور پر درجہ بندی کرنے کے لیے کئی شرطوں کی جانچ کی جاتی ہے۔

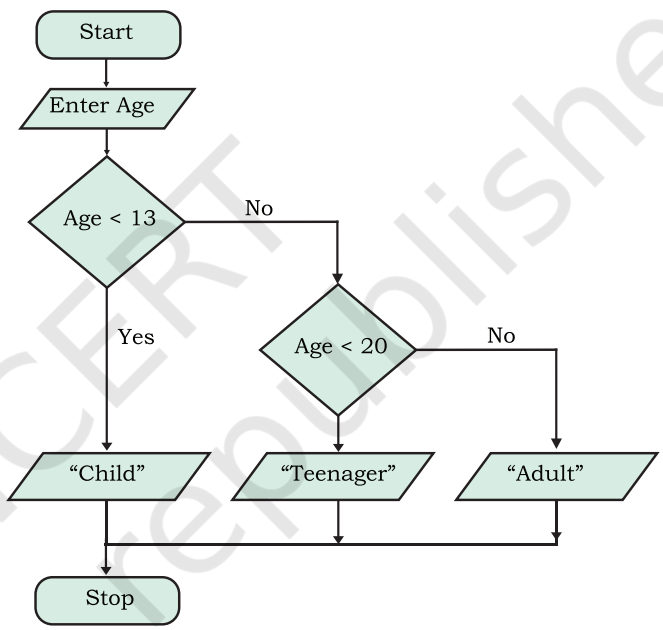
- ان پُٹ: عمر
- پروسیس (عمل): دیے ہوئے معیار کی بنیاد پر عمر کی جانچ کیجیے
- پرنٹ کیجیے ''بچہ'' یا ''نوجوان'' یا ''بالغ''

نوٹ

سوڈوکوڈ مندرجہ ذیل ہے:

```
INPUT Age
IF Age < 13 THEN
       PRINT Child
  ELSE IF Age < 20 THEN
       PRINT Teenager
  ELSE
       PRINT Adult
```

الگورتھم کی نمائندگی کرنے والے فلو چارٹ کو شکل 4.9 میں دکھایا گیا ہے۔



شکل 9.4 : کئی شرائط کی جانچ کرنے کے لیے فلو چارٹ

مثال 7.4: "Dragons and Wizards" نامی کارڈ گیم کے لیے الگورتھم

DRAGONS اور WIZARDS دو ٹیمیں بنائیے۔

گیم کے ضابطے ذیل میں دیے گئے ہیں۔

- اگر نکالا گیا کارڈ اینٹ یا چڑی ہے تو ٹیم Dragons کو ایک پوائنٹ ملے گا۔
- اگر نکالا گیا کارڈ پان ہے جس پر کوئی عدد لکھا ہوا ہے تو ٹیم Wizards کو ایک پوائنٹ ملے گا۔
- اگر نکالا گیا کارڈ پان ہے جس پر کوئی عدد نہیں لکھا ہوا ہے تو ٹیم Dragons کو ایک پوائنٹ ملے گا۔
- اگر نکالا گیا کارڈ کوئی اور ہے تو ٹیم Wizards کو ایک پوائنٹ ملے گا۔
- سب سے زیادہ پوائنٹ حاصل کرنے والی ٹیم فاتح قرار دی جائے گی۔

آیئے ایک کارڈ کے لیے مندرجہ ذیل کی شناخت کریں:

ان پُٹ: شکل، قدر

نوٹ

پراسیس (عمل): نکالے گئے کارڈ کے نتیجے کی بنیاد پر متعلقہ ٹیم کے اسکور میں عدد ایک کا اضافہ، جیسا کہ گیم کے ضابطوں میں وضاحت کی گئی ہے۔

آؤٹ پُٹ: فاتح ٹیم

آیئے اب گیم کے لیے مشروط بیانات تحریر کریں:

```
IF (shape is diamond) OR (shape is club)
     Team DRAGONS gets a point
ELSE IF (shape is heart) AND (value is
number)
     Team WIZARDS gets a point
ELSE IF (shape is heart) AND (value is not a
number)
     Team DRAGONS gets a point
ELSE
     Team WIZARDS gets a point
```

پروگرام کا سوڈو کوڈ مندرجہ ذیل ہوسکتا ہے:

نوٹ: Dpoint (Dragon کے لیے) اور Wpoint (Wizard کے لیے) متعلقہ ٹیموں کے ذریعے حاصل کیے گئے پوائنٹ کو اسٹور کرتے ہیں۔

```
INPUT shape
INPUT value
SET Dpoint = 0, Wpoint = 0
IF (shape is diamond) OR (shape is club) THEN
     INCREMENT Dpoint
ELSE IF (shape is heart) AND (value is
number) THEN
     INCREMENT Wpoint
ELSE IF (shape is heart) AND (value is not a
number) THEN
     INCREMENT Dpoint
ELSE
     INCREMENT Wpoint
END IF
IF Dpoint > Wpoint THEN
  PRINT "Dragon team is the winner"
ELSE
  PRINT "Wizard team is the winner"
```

4.5.3 تکرار (Repetition)

کسی مقام پر جانے کی ہدایات دیتے وقت ہم کچھ اس طرح کے جملوں کا استعمال کرتے ہیں مثلاً ''50 قدم چلیے اور پھر دائیں مڑیے'' یا ''اگلے چوراہے تک چلتے جائیں اور پھر دائیں مڑیں''۔ کچھ اور مثالوں پر غور کریں مثلاً

- پانچ مرتبہ تالی بجائیں
- 10 قدم آگے بڑھیں
- اسی جگہ کودتے رہیں جب تک تھک نہ جائیں

اس قسم کے بیانات کو ہم اس وقت استعمال کرتے ہیں جب ہم یہ چاہتے ہیں کہ کسی کام کو ایک مقررہ تعداد میں بار بار انجام دیا جائے۔ اسی طرح، فرض کیجیے کہ پچھلے والے کارڈ گیم (مثال 4.7) میں 10 کارڈ نکالے جانے ہیں، تو جیتنے والی ٹیم کا فیصلہ کرنے کے لیے سوڈوکوڈ کو 10 مرتبہ دہرانا ہوگا۔ یہ سبھی تکرار (Repetitions) کی مثالیں ہیں۔ پروگرامنگ میں تکرار کو اعادہ (Iteration) یا لوپ (Loop) کہا جاتا ہے۔ الگورتھم میں لوپ کا مطلب ہے کہ کسی متعینہ شرط کے پورا ہونے تک پروگرام کے کچھ بیانات کو بار بار اگزیکیوٹ کرنا۔

مثال 4.8: 5 اعداد کو حاصل کرنے اور ان کا اوسط معلوم کرنے کے لیے سوڈوکوڈ لکھیے اور فلوچارٹ بنائیے۔

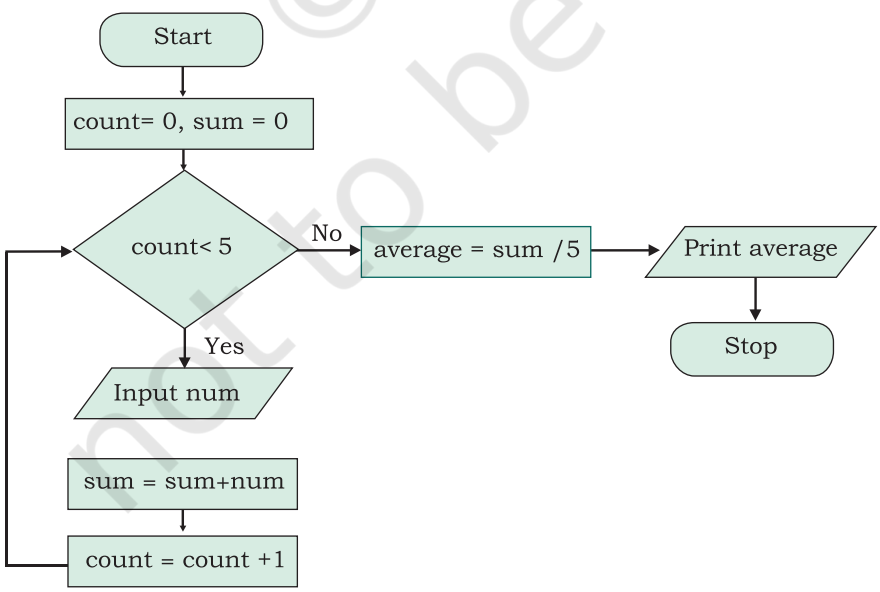
فلوچارٹ کو شکل 4.10 میں دکھایا گیا ہے۔

سوڈوکوڈ مندرجہ ذیل ہوگا:

```
Step 1: Set count = 0, sum = 0
Step 2: While count <5 , repeat steps 3 to 5
Step 3: Input a number to num
Step 4: sum = sum + num
Step 5: count = count + 1
Step 6: Compute average = sum/5
Step 7: Print average
```

مثال 4.8 میں "Count" نام کا ایک کاؤنٹر لوپ کی تکرار کی تعداد کا حساب رکھتا ہے۔ لوپ کی ہر ایک تکرار کے بعد Count کی قدر میں اس وقت تک 1 کا اضافہ ہوتا جاتا ہے جب تک کہ اس متعینہ تعداد میں تکراری عمل کو انجام نہیں دے دیتا جس تعداد کو اعادہ کی شرط میں دیا گیا ہے۔

ایسے بھی صورت حال پیدا ہوجاتی ہے جب ہمیں پہلے سے یہ معلوم نہیں ہوتا کہ بیانات کے سیٹ کو کتنی مرتبہ دہرایا جانا ہے۔ نامعلوم تعداد میں تکرار کے ان عملوں کو While کی مدد سے انجام دیا جاتا ہے۔



شکل 4.10 : پانچ اعداد کے اوسط کے تحسیب کے لیے فلوچارٹ

سوچیے اور جواب دیجیے

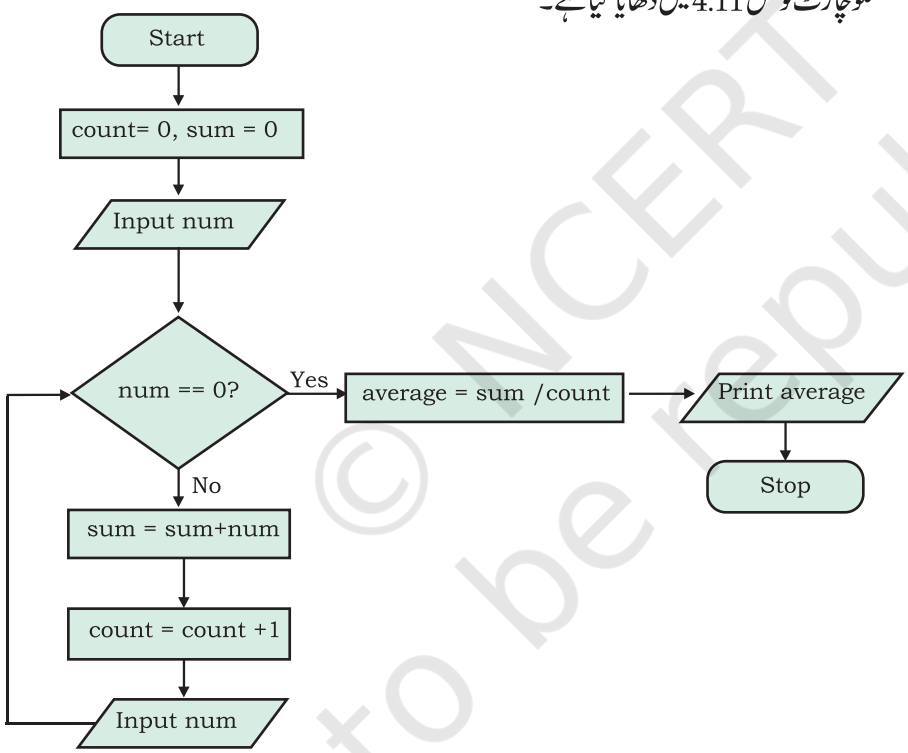
کیا آپ اپنی روزمرہ زندگی کے معمول سے متعلق ایسی سرگرمیوں کی فہرست بنا سکتے ہیں جن میں تکرار یا اعادہ شامل ہو۔

مثال 4.9 ایک ایسے پروگرام کا سوڈوکوڈ لکھیے اور فلو چارٹ بنائیے جو استعمال کنندہ کے ذریعے 0 داخل کیے جانے تک اعداد کو حاصل کرتا ہے اور اس کے بعد ان کے اوسط کی تحسیب کرتا ہے۔

سوڈوکوڈ مندرجہ ذیل ہے:

Step 1: Set count = 0, sum = 0
Step 2: Input num
Step 3: While num is not equal to 0, repeat Steps 4 to 6
Step 4: sum = sum + num
Step 5: count = count + 1
Step 6: Input num
Step 7: Compute average = sum/count
Step 8: Print average

فلو چارٹ کو شکل 4.11 میں دکھایا گیا ہے۔



شکل 4.11: استعمال کنندہ کے ذریعے 0 داخل کیے جانے تک اعداد کو حاصل کرنے کے لیے فلو چارٹ

اس مثال میں ہمیں یہ معلوم نہیں ہے کہ استعمال کنندہ صفر (0) داخل کرنے سے پہلے کتنے اعداد داخل کرے گا۔ اس معاملے میں شرط کی اس وقت تک مکرر انداز میں جانچ کی جاتی ہے جب تک وہ False نہیں ہو جاتی۔

4.6 الگورتھم کی تصدیق کرنا (VERIFYING ALGORITHMS)

ذرا سوچیے، اگر ایک بینکنگ سافٹ ویئر صحیح طریقے سے کام نہ کر پائے تو کیا ہوگا؟ فرض کیجیے کہ آن لائن منی

سرگرمی 4.4

مثال 4.9 میں دیے ہوئے سوڈوکوڈ کا استعمال کر کے آیئے مندرجہ ذیل سوالوں کے جواب دیجیے۔

1) اگر ان پُٹ 6, 7, 4, 8, 2, 5, 0, 3, 1 ہے تو حاصل جمع کیا ہوگا؟
2) گنتی کی قدر کیا ہوگی؟
3) ہم num کو دو مرتبہ داخل کرنے کے لیے ان پُٹ بیان کا استعمال کیوں کرتے ہیں؟
4) ہم حاصل جمع کو گنتی سے تقسیم کیوں کرتے ہیں؟
5) کیا کوئی اور طریقہ بھی ہو سکتا ہے؟

ٹرانسفر ماڈیول کے کام کاج کو صحیح طریقے سے پروگرام نہیں کیا گیا ہے اور یہ کھاتے میں منتقل کی گئی رقم کا صرف نصف حصہ ہی جمع کرتا ہے! اگر کھاتے میں رقم جمع ہونے (Credit) کے بجائے نکل جائے (Debit) تو کیا ہوگا۔ اس قسم کا ناقص سافٹ ویئر پورے سسٹم کی کارکردگی کو بگاڑ دے گا اور تباہی مچا دے گا! آج سافٹ ویئر کا استعمال اس سے بھی زیادہ اہم خدمات کی انجام دہی میں کیا جاتا ہے مثلاً شعبہ طب یا خلائی شٹل وغیرہ۔ ایسے سافٹ ویئر کے لیے لازم ہے کہ وہ ہر صورت میں بالکل صحیح طریقے سے کام کریں۔ لہٰذا، سافٹ ویئر تیار کرنے والے افراد کے لیے اس بات کو یقینی بنانا ضروری ہے کہ سبھی اجزا کی فعلیت کو بالکل صحیح طریقے سے متعین کیا گیا ہے اور ہر ممکنہ انداز میں اس کی جانچ اور تصدیق کی گئی ہے۔

سوچیے اور جواب دیجیے

مسئلے کو حل کرنے کے معاملے میں الگورتھم کی تصدیق اہم کیوں ہے؟

جب ہمیں یہ بتایا گیا کہ پہلے N فطری اعداد کا حاصل جمع معلوم کرنے کا فارمولا $\frac{N(N+1)}{2}$ ہے تو ہم نے اس کی تصدیق کس طرح کی؟ ہم چھوٹے اعداد لے کر اس کی جانچ کر سکتے ہیں۔ اس کے لیے ہم اعداد کا حاصل جمع معلوم کریں گے۔ فرض کیجیے کہ N=6 ہے، لہٰذا حاصل جمع 1 + 2 + 3 + 4 + 5 + 6 یعنی 21 ہوگا۔

فارمولا استعمال کرنے پر حاصل جمع = $\frac{6x(6+1)}{2}$

ہم اسی طرح کچھ اور اعداد لے کر اس کی جانچ کر سکتے ہیں اور اس بات کو یقینی بنا سکتے ہیں کہ فارمولا صحیح طریقے سے کام کر رہا ہے۔

اسی طرح جب ہم الگورتھم لکھتے ہیں تو ہم اس بات کی تصدیق کرنا چاہتے ہیں کہ یہ توقع کے مطابق کام کر رہا ہے یا نہیں۔ تصدیق کرنے کے لیے ہمیں مختلف ان پُٹ قدریں لینی ہوں گی اور ہر ایک ان پُٹ قدر کے لیے مطلوبہ آؤٹ پُٹ حاصل کرنے کے مقصد سے الگورتھم کے سبھی مراحل سے گزرنا ہوگا اور ضرورت کے مطابق اس میں ترمیم یا اصلاح کی جا سکتی ہے۔ ان پُٹ حاصل کرنے اور الگورتھم کے مراحل سے ہو کر گزرنے کو بعض اوقات 'ڈرائی رن' (Dry Run) کہتے ہیں۔ اس قسم کے ڈرائی رن سے ہمیں مندرجہ ذیل میں مدد ملتی ہے۔

1۔ الگورتھم میں غلط مراحل کی شناخت کرنا

2۔ الگورتھم میں گمشدہ تفصیلات یا تخصیصات کو معلوم کرنا

سیمولیشن کے لیے استعمال کیے جانے والی ان پُٹ قدر کی قسم کا تعین بہت اہم ہے۔ اگر سبھی ممکنہ ان پُٹ قدروں کی جانچ نہیں کی جاتی ہے تو پروگرام ناکام ہو جائے گا۔ اگر کوئی اور ایسا معاملہ ہے جس میں یہ الگورتھم کام نہیں کرتا ہے تو کیا ہوگا؟ آیئے کچھ مثالوں پر غور کریں۔

سرگرمی 4.5

ایک مستطیل نما شکل کی لمبائی اور چوڑائی کی پیمائش کو فٹ اور انچ میں (مثلاً 5 فٹ 6 انچ) ان پُٹ کے طور پر لینے کے لیے ایک الگورتھم لکھیے۔ اس شکل کا رقبہ اور احاطہ بھی معلوم کیجیے۔

مقام B سے ہو کر A سے C (T_total) تک جانے میں لگنے والے وقت کی تخمیب کے لیے ایک الگورتھم لکھیے، جہاں A سے B (T1) تک اور B سے C (T2) تک جانے میں لگنے والا وقت دیا گیا ہے۔

نوٹ

یعنی ہم چاہتے ہیں کہ الگورتھم گھنٹہ اور منٹ میں دیے گئے وقت کو جمع کرے۔ الگورتھم کو لکھنے کا ایک طریقہ مندرجہ ذیل ہے:

```
PRINT value for T1
INPUT hh1
INPUT mm1
PRINT value for T2
INPUT hh2
INPUT mm2
hh_total = hh1 + hh2 (Add hours)
mm_total = mm1 + mm2 (Add mins)
Print T_total as hh_total, mm_total
```

آیئے، اب اس کی تصدیق کریں۔ فرض کیجیے کہ پہلی مثال جس پر ہم غور کرتے ہیں اس میں T1 = 5 گھنٹے 20 منٹ اور T2 = 7 گھنٹے 30 منٹ۔ ڈرائی رن کرنے پر ہمیں نتیجے کے طور پر 12 گھنٹے 50 منٹ حاصل ہوتے ہیں۔ یہ صحیح معلوم ہوتا ہے۔

اب ہم ایک اور مثال لیتے ہیں جہاں T1 = 4 گھنٹے 50 منٹ اور T2 = 2 گھنٹے 20 منٹ ہے، ہمیں نتیجے کے طور پر 6 گھنٹے 70 منٹ حاصل ہوتے ہیں جو کہ وقت کی پیمائش کا صحیح طریقہ نہیں ہے۔ نتیجہ 7 گھنٹے 10 منٹ ہونا چاہیے تھا۔

اس دوسری مثال میں ہم یہ محسوس کرتے ہیں کہ ہمارا الگورتھم اسی وقت کام کرے گا جب mm1 + mm2 (mm_total) < 60 ہے۔ دیگر سبھی معاملوں میں ہمیں وہ آؤٹ پُٹ نہیں ملے گا جیسا ہم چاہتے ہیں۔ جب mm_total >= 60 ہے تو الگورتھم کو گھنٹوں کے حاصل جمع (hh_total) میں 1 کا اضافہ اور mm_total کی قدر میں 60 کی کمی کر دینی چاہیے یعنی (mm_total-60)۔ چنانچہ ترمیم شدہ الگورتھم مندرجہ ذیل ہوگا:

```
PRINT value for T1
INPUT hh1
INPUT mm1
PRINT value for T2
INPUT hh2
INPUT mm2
hh_total = hh1 + hh2 (Add hours)
mm_total = mm1 + mm2 (Add mins)
hh_total = hh1 + hh2 (Add hours)
mm_total = mm1 + mm2 (Add mins)
IF (mm_total >= 60) THEN
     hh_total = hh_total + 1
     mm_total = mm_total - 60
PRINT T_total as hh_total, mm_total
```

نوٹ

اب ہم الگورتھم کے ذریعے T1=4 گھنٹے 50 منٹ اور T2=2 گھنٹے 20 منٹ کے لیے سیمولیٹ کرسکتے ہیں اور 7 گھنٹے 10 منٹ حاصل کرسکتے ہیں جس کا مطلب ہے کہ الگورتھم صحیح طریقے سے کام کررہا ہے۔

فرض کیجیے کہ ہم زیرغور الگورتھم کی تصدیق کیے بغیر کوئی سافٹ ویئر تیار کرلیتے ہیں اور اگر الگورتھم میں غلطیاں موجود ہیں تو یہ سافٹ ویئر کام نہیں کرے گا۔ چنانچہ الگورتھم کی تصدیق کرنا بہت اہم ہے کیوں کہ غلطی کو پکڑنے اور اس کی تصحیح کے لیے درکار کوشش کم سے کم ہوجاتی ہے۔

4.7 الگورتھم کا موازنہ (COMPARISON OF ALGORITHM)

کمپیوٹر کا استعمال کرکے کسی مسئلے کو حل کرنے کے کئی طریقے ہوسکتے ہیں لہٰذا ہمارے پاس ایک سے زیادہ الگورتھم ہوسکتے ہیں۔ اب سوال یہ اٹھتا ہے کہ کون سے الگورتھم کو استعمال کرنا چاہیے؟

ایک ایسے مسئلے پر غور کیجیے جس میں یہ معلوم کرنا ہے کہ آیا دیا ہوا عدد مفرد ہے یا نہیں۔ کمپیوٹر سائنس میں مفرد اعداد انتہائی اہمیت کے حامل ہیں کیوں کہ ان کا اطلاق ڈیٹا بیس، سکیورٹی، فائل کمپریشن، ڈی کمپریشن (Decompression)، ماڈیولیشن اور ڈی ماڈیولیشن (Demodulation) وغیرہ میں کیا جاتا ہے۔ دیا ہوا عدد مفرد ہے یا نہیں، اس بات کی جانچ کرنے کے لیے الگورتھم لکھنے کے چار طریقے ہوسکتے ہیں جیسا کہ ذیل میں دکھایا گیا ہے۔

(i) قاسم 2 سے شروع کرتے ہوئے دیے ہوئے عدد (مقسوم) کو تقسیم کریں اور اس بات کی جانچ کریں کہ کیا کوئی جزوضربی ہے۔ ہر ایک اعادہ کے بعد قاسم میں اضافہ کریں اور پچھلے مراحل کو اس وقت تک دہرائیں جب تک کہ مقسوم > قاسم نہ ہوجائے۔ اگر کوئی جزوضربی ہے تو دیا ہوا عدد مفرد نہیں ہے۔

(ii) (i) میں، مقسوم تک سبھی اعداد کو آزمانے کے بجائے، دی ہوئی قدر (مقسوم) کے نصف تک ہی آزمایئے کیوں کہ قاسم مقسوم کے نصف سے زیادہ نہیں ہوسکتا۔

(iii) طریقہ (i) میں، مقسوم (اعداد) کے صرف جذرالمربع تک ہی آزمایئے۔

(iv) 100 تک مفرد اعداد کی پہلے سے ایک فہرست دی ہوئی ہے۔ دیے ہوئے عدد کو فہرست کے ہر ایک عدد سے تقسیم کیجیے۔ اگر کسی بھی عدد سے قابل تقسیم نہیں ہے تو عدد مفرد ہے بصورت دیگر مفرد نہیں ہے۔

یہ سبھی چاروں طریقے اس بات کی جانچ کرسکتے ہیں کہ آیا دیا ہوا عدد مفرد ہے یا نہیں۔ لیکن اب سوال یہ ہے کہ ان میں سے کون سا طریقہ بہتر یا کارگر ہے؟

تحلیل کے ذریعے کسی مسئلے کو حل کرنے کا رجحان 'تقسیم اور فتح' کی مانند ہے۔ ایک مشہور ریاضی داں ہاورڈ رفا کے الفاظ میں:

''ایک پیچیدہ مسئلے کو سادہ مسئلوں کی شکل میں تحلیل کریں۔ ان سادہ مسائل پر اپنی فکر کا اطلاق کریں۔ ان تجزیوں کو منطقی گوند کی مدد سے یکجا کریں''۔

الگورتھم (i) میں بڑی تعداد میں تحسیبات کرنی ہوں گی، اس کا مطلب ہے کہ پروسیسنگ کے لیے زیادہ وقت درکار ہوگا کیوں کہ جب تک قاسم عدد سے کم ہے یہ سبھی اعداد کی جانچ کرے گا۔ اگر دیا ہوا عدد بڑا ہے تو یہ طریقہ آؤٹ پُٹ دینے میں زیادہ وقت لے گا۔

الگورتھم (i) کے مقابلے میں (ii) زیادہ کارگر ہے کیوں کہ یہ عدد کے نصف تک ہی تقسیم پذیری کی جانچ کرتا ہے اور اس طرح مفرد عدد کی تحسیب میں لگنے والے وقت کو کم کرتا ہے۔ الگورتھم (iii) اور بھی زیادہ کارگر ہے کیوں کہ یہ عدد کے جذر المربع تک ہی تقسیم پذیری کی جانچ کرتا ہے چنانچہ پروسیسنگ میں اور بھی کم وقت لگتا ہے۔

الگورتھم (iv) تقسیم پذیری کے لیے صرف ایسے مفرد اعداد کا استعمال کرتا ہے جو دیے ہوئے عدد سے چھوٹے ہیں یہ تحسیبات کو مزید کم کر دیتا ہے۔ لیکن اس طریقے میں، پہلے ہمیں مفرد اعداد کی فہرست اسٹور کرنے کی ضرورت ہوگی چنانچہ کم تحسیبات ہونے کے باوجود اس کے لیے اضافی میموری درکار ہوگی۔

لہٰذا، الگورتھم کا موازنہ اور تجزیہ انھیں چلانے کے لیے درکار پروسیسنگ مدت اور الگورتھم کو ایگزیکیوٹ کرنے کے لیے درکار میموری کی مقدار کی بنیاد پر کیا جاسکتا ہے۔ انھیں بالترتیب زمانی پیچیدگی (Time complexity) اور مکانی پیچیدگی (Space complexity) کہا جاتا ہے۔

کسی الگورتھم کو دوسرے الگورتھم کے مقابلے منتخب کیا جانا اس بات پر منحصر ہوتا ہے کہ وہ درکار پروسیسنگ مدت (زمانی پیچیدگی) اور استعمال کی جانے والی میموری (مکانی پیچیدگی) کے معاملے میں کتنا کارگر ہے۔

4.8 کوڈنگ (CODING)

ایک مرتبہ الگورتھم کو حتمی شکل دینے کے بعد پروگرامر کے ذریعے منتخب کی گئی ہائی لیول پروگرامنگ لینگویج میں اس کی کوڈنگ (کوڈ کی شکل میں تحریر کرنا) کی جاتی ہے۔ ہدایات کے ایک مرتب سیٹ کو منتخب کی گئی پروگرامنگ لینگویج میں اس کی نحوی ترکیب کو استعمال کر کے لکھا جاتا ہے۔ نحوی ترکیب ضوابط یا قواعد کا ایسا سیٹ ہے جو زبان میں بیانات کی تشکیل کو کنٹرول کرتا ہے مثلاً املا، رموز اوقاف، الفاظ کی ترتیب وغیرہ۔

مشینی زبان (ML) یا ادنیٰ سطحی زبان (LLL) جو صرف 0s اور 1s پر مشتمل ہے کمپیوٹر پروگرام کو لکھنے کا ایک مثالی طریقہ ہے۔ بائنری ڈجٹ کا استعمال کر کے تحریر کیے گئے پروگرام کو کمپیوٹر ہارڈ ویئر براہ راست سمجھ سکتا ہے لیکن انسان انھیں سمجھنے یا استعمال کرنے سے قاصر ہوتے ہیں۔ اعلیٰ سطحی زبانوں کی دریافت اسی وجہ سے ہوئی جو فطری زبانوں کے بہت قریب ہیں اور انھیں پڑھنا، لکھنا آسان ہے لیکن کمپیوٹر ہارڈ ویئر ان زبانوں کو براہ راست نہیں سمجھ پاتا ہے۔ اعلیٰ سطحی زبانوں کو استعمال کرنے کا ایک فائدہ یہ ہے کہ وہ قابل منتقل (پورٹیبل) ہوتی ہیں یعنی انھیں معمولی یا بغیر ترمیم کے مختلف قسم کے کمپیوٹروں پر چلایا جاسکتا ہے۔ ادنیٰ سطحی پروگراموں کو صرف ایک ہی قسم کے کمپیوٹروں پر چلایا جاسکتا ہے اور دوسرے قسم کے سسٹم پر چلانے کے لیے

نوٹ

انھیں دوبارہ تحریر کرنا پڑتا ہے۔اعلیٰ سطحی زبانوں میں FORTRAN، C، C++، Java، Python وغیرہ شامل ہیں۔

اعلیٰ سطحی زبان میں لکھے گئے پروگرام کو سورس کوڈ کہتے ہیں۔ہم کمپائلر یا انٹرپریٹر کا استعمال کر کے سورس کوڈ کا مشینی زبان میں ترجمہ کرتے ہیں تا کہ کمپیوٹر اسے سمجھ سکے۔ہم کمپائلر اور انٹرپریٹر کے بارے میں باب 1 میں پڑھ چکے ہیں۔

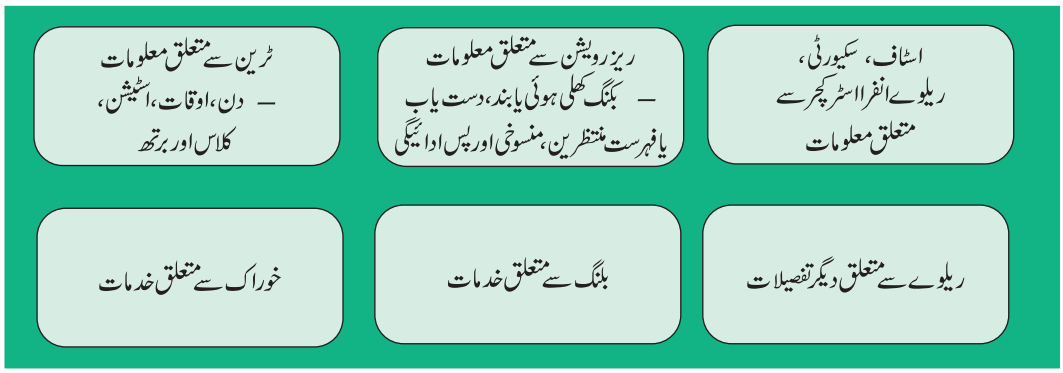
پروگرامنگ کے لیے متعدد زبانیں دست یاب ہیں۔اپنی ضروریات کے پیش نظر کسی ایک مناسب زبان کا انتخاب کرنے کے لیے ہمیں کئی عوامل پر غور کرنا ہوگا۔اس کا انحصار اس پلیٹ فارم (OS) پر ہے جہاں پروگرام چلے گا۔ہمیں یہ طے کرنا ہے کہ آیا ایپلی کیشن ایک ڈیسک ٹاپ ایپلی کیشن ہے، موبائل ایپلی کیشن ہے یا ویب ایپلی کیشن ہے۔ڈیسک ٹاپ اور موبائل ایپلی کیشن کو عام طور پر ایک مخصوص آپریٹنگ سسٹم اور کچھ ہارڈ ویئر کے لیے تیار کیا جاتا ہے جب کہ ویب ایپلی کیشن کو ویب براؤزر کی مدد سے مختلف ڈیوائسیز پر ایکسس کیا جاسکتا ہے اور یہ کلاؤڈ پر دست یاب وسائل کا استعمال کر سکتی ہیں۔

علاوہ ازیں، پروگرام نہ صرف کمپیوٹر، موبائل یا ویب براؤزر پر کام کرنے کے لیے تیار کیے جاتے ہیں بلکہ انھیں ڈیجیٹل واچ، mp3 پلیئر، ٹریفک سگنل یا گاڑیوں، طبی آلات اور دیگر اسمارٹ ڈیوائسز جیسے نصب شدہ نظاموں (Embedded Systems) کے لیے بھی تحریر کیا جاسکتا ہے۔ان معاملوں میں ہمیں دیگر خصوصی پروگرامنگ ٹول کو استعمال کرنا پڑتا ہے یا بعض اوقات پروگراموں کو اسمبلی لینگویج میں تحریر کیا جاتا ہے۔

4.9 تحلیل (DECOMPOSITION)

بعض اوقات کوئی مسئلہ بہت زیادہ پیچیدہ بھی ہوسکتا ہے یعنی اس کا حل براہ راست قابل حصول نہیں ہوتا۔ایسے معاملوں میں، ہمیں اسے سادہ حصوں میں تحلیل کرنے کی ضرورت پیش آتی ہے۔آیئے ریلوے ریزرویشن سسٹم پر غور کریں جس کے بارے میں ہم پہلے بات کر چکے ہیں۔ایک اچھے ریلوے ریزرویشن سسٹم کو ڈیزائن کرنے کے پیچیدہ کام کو ہم سسٹم کے مختلف اجزا کو ڈیزائن کرنے اور پھر ایک دوسرے کے ساتھ موثر انداز میں کام کرنے والے نظام کے طور پر دیکھ سکتے ہیں۔

ایک پیچیدہ مسئلہ کو تحلیل کے ذریعے حل کرنے کا بنیادی تصور یہ ہے کہ پیچیدہ مسئلے کو نسبتاً چھوٹے ذیلی مسئلوں میں تحلیل یا تقسیم کر دیا جاتا ہے جیسا کہ شکل 4.12 میں دکھایا گیا ہے۔اصل مسئلے کے مقابلے میں ان ذیلی مسائل کو حل کرنا نسبتاً آسان ہوتا ہے۔آخر میں، اصل مسئلے کا حل معلوم کرنے کے لیے ذیلی مسئلوں کو منطقی انداز میں ایک دوسرے کے ساتھ جوڑ دیا جاتا ہے۔



شکل 4.12 : ریلوے ریزرویشن سسٹم

ایک پیچیدہ مسئلے کو ذیلی مسئلوں میں توڑنے یا تقسیم کرنے کا مطلب یہ بھی ہے کہ ہر ایک ذیلی مسئلے کی تفصیل سے جانچ کی جاسکتی ہے۔ ہر ایک ذیلی مسئلے کو آزادانہ طور پر اور مختلف افراد (ٹیم) کے ذریعے حل کیا جاسکتا ہے۔ مختلف ذیلی مسئلوں پر کام کرنے کے لیے مختلف ٹیموں کے ہونے کے فائدے بھی ہیں کیوں کہ مخصوص ذیلی مسئلے ایسی ٹیموں کو تفویض کیے جاسکتے ہیں جنھیں اس قسم کے مسئلوں کے حل کرنے میں مہارت حاصل ہے۔

حقیقی زندگی سے وابستہ ایسے کئی مسائل ہیں جنھیں تحلیل کی مدد سے حل کیا جاسکتا ہے۔ اس قسم کی مثالوں میں ریاضی اور سائنس کے مسئلوں کو حل کرنا، اسکول میں کسی تقریب کا بندوبست اور انتظام، موسم کی پیشین گوئی، ڈلیوری مینجمنٹ سسٹم وغیرہ شامل ہیں۔

انفرادی ذیلی مسئلوں کو حل کرنے کے بعد یہ ضروری ہے کہ ان کی درستگی کی جانچ کی جائے اور مکمل حل حاصل کرنے کے لیے انھیں باہم مربوط کیا جائے۔

خلاصہ

- الگورتھم کی تعریف کسی کام کو انجام دینے کے لیے ڈیزائن کیے گئے مرحلہ وار طریق کار کے طور پر کی جاتی ہے۔ اگر صحیح طریقے سے عمل کیا جائے تو یہ ایک متناہی مدت میں مطلوبہ نتیجہ فراہم کرے گا۔
- الگورتھم کی ایک متعین ابتدا اور انتہا ہوتی ہے اور اس کے مراحل کی تعداد بھی متناہی ہوتی ہے۔
- ایک اچھا الگورتھم جو بالکل درست، منفرد اور متناہی ہے، ان پُٹ حاصل کرتا ہے اور آؤٹ پُٹ فراہم کرتا ہے۔
- ایک موثر الگورتھم لکھنے کے لیے ہمیں ان پُٹ، اختیار کیے جانے والے طریقے اور مطلوبہ آؤٹ پُٹ کی شناخت کرنی ہوگی۔
- فلوچارٹ ایک قسم کا ڈائیگرام ہے جو تیر کے نشان کی مدد سے باہم مربوط مختلف قسم کے باکس کا استعمال کرکے الگورتھم کو گراف کی شکل میں پیش کرتا ہے۔

نوٹ

- ایسے الگورتھم جہاں سبھی مراحل یکے بعد دیگرے ایگزیکیوٹ ہوتے ہیں تو یہ کہا جاتا ہے کہ ان کا ایگزیکیوشن تسلسل میں ہے۔
- فیصلہ سازی میں شرط کے نتیجے کی بنیاد پر کسی ایک متبادل کا انتخاب شامل ہے۔
- الگورتھم میں مراحل کا ایک ایسا مخصوص سیٹ بھی ہوسکتا ہے جنھیں ایک متناہی تعداد میں دہرایا جارہا ہو۔اس قسم کے الگورتھم کو تکراری کہا جاتا ہے۔
- کسی مسئلے کو حل کرنے کے کئی طریقے ہوسکتے ہیں اور اسی لیے ہمارے پاس ایک زیادہ الگورتھم ہوسکتے ہیں۔
- الگورتھم کا انتخاب زمانی اور مکانی پیچیدگی کی بنیاد پر کیا جانا چاہیے۔

مشق

1۔ ایک ایسا سوڈوکوڈ لکھیے جو دو اعداد کو پڑھتا ہے اور ایک عدد کو دوسرے عدد سے تقسیم کرتا ہے اور خارج قسمت کو ظاہر کرتا ہے۔

2۔ دو دوستوں نے ایک سکے کو پانچ مرتبہ اچھال کر یہ طے کیا کہ کیک کا آخری ٹکڑا کسے ملے گا۔ تین فلپ (اچھال) جیتنے والا پہلا شخص کیک جیت جائے گا۔ 1 ان پُٹ کا مطلب ہے کہ کھلاڑی 1 ایک فلپ جیت جاتا ہے اور 2 کا مطلب ہے کہ کھلاڑی 2 ایک فلپ جیت جاتا ہے۔ یہ متعین کرنے کے لیے کہ کیک کس کو ملے گا ایک الگورتھم لکھیے۔

3۔ 10 اور 25 (10 اور 25 دونوں شامل ہیں) کے درمیان 5 کے سبھی اضعاف کو پرنٹ کرنے کے لیے ایک سوڈوکوڈ لکھیے۔

4۔ ایک ایسے لوپ کی مثال دیجیے جسے ایک مقررہ تعداد میں دوہرایا جائے گا۔

5۔ فرض کیجیے کہ آپ کسی کام کے لیے رقم جمع کر رہے ہیں۔ آپ کو کل 200 روپے چاہئیں۔ آپ اپنے والدین، چچا چچی اور دادا دادی سے مانگ سکتے ہیں۔ لوگ آپ کو 10 روپے، 20 روپے یا 50 روپے بھی دے سکتے ہیں۔ آپ اس وقت تک جمع کریں گے جب تک کہ کل رقم 200 نہ ہو جائے۔ الگورتھم لکھیے۔

6۔ کسی شے کی قیمت اور مقدار کی بنیاد پر بل کو پرنٹ کرنے کے لیے سوڈوکوڈ لکھیے۔ بل جی ایس ٹی بھی پرنٹ کیجیے جو کُل بل میں 5% ٹیکس جوڑنے کے بعد حاصل ہونے والا بل ہے۔

7۔ ایک سوڈوکوڈ لکھیے جو مندرجہ ذیل کاموں کو انجام دیتا ہے:

a) تین مضامین یعنی کمپیوٹر سائنس، ریاضی اور طبیعیات کے نمبروں (100 میں سے) کو پڑھتا ہے۔

نوٹ

b) کل نمبروں کی تحسیب کرتا ہے۔

c) فیصد نمبروں کی تحسیب کرتا ہے۔

8۔ استعمال کنندہ کے ذریعے داخل کیے گئے دو مختلف اعداد میں سے سب سے بڑا عدد معلوم کرنے کے لیے ایک الگورتھم لکھیے۔

9۔ ایک الگورتھم لکھیے جو مندرجہ ذیل کاموں کو انجام دیتا ہے:

استعمال کنندہ سے کوئی عدد داخل کرنے کے لیے کہتا ہے۔ اگر وہ عدد 5 اور 15 کے درمیان میں ہے تو لفظ GREEN لکھتا ہے۔ اگر عدد 15 اور 25 کے درمیان میں ہے تو لفظ BLUE لکھتا ہے۔ اگر عدد 25 اور 35 کے درمیان میں ہے تو لفظ ORANGE لکھتا ہے۔ اگر یہ کوئی اور عدد ہے تو یہ ALL COLOURS ARE BEAUTIFUL لکھتا ہے۔

10۔ ایک پروگرام لکھیے جو ان پُٹ کے طور پر چار اعداد کو حاصل کرتا ہے اور ان میں سے سب سے بڑے اور سب سے چھوٹے عدد کی نشاندہی کرتا ہے۔

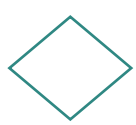
11۔ ایک الگورتھم لکھیے جو مندرجہ ذیل شرح کے مطابق گاہک کے ذریعے صرف کی گئی یونٹ کی تعداد کی بنیاد پر ایک ماہ کے پانی کے بل کی کل رقم کو ظاہر کرتا ہے۔

- پہلی 100 یونٹ کے لیے @5 فی یونٹ
- اگلی 150 یونٹ کے لیے @10 فی یونٹ
- 250 سے زیادہ یونٹ کے لیے @20 فی یونٹ

پانی کے بل کی کل رقم کا حساب لگانے کے لیے اس میں فی ماہ 75 روپے میٹر چارج بھی جمع کیجیے۔

12۔ کنڈیشنل کیا ہیں؟ پروگرام میں ان کی ضرورت کب پڑتی ہے؟

13۔ جوڑے ملائیے۔

فلوچارٹ کی علامات	فنکشن
	کنٹرول کا بہاؤ
	پروسیس کا مرحلہ
	پروسیس کی ابتدا اور اختتام
	ڈیٹا
	فیصلہ سازی

نوٹ

14۔ اسکول یا کالج جانے کے لیے الگورتھم ذیل میں دیا گیا ہے۔ کیا آپ دیگر متبادلات کو شامل کرنے کے لیے اس میں کسی قسم کی ترمیم تجویز کریں گے؟

Reach_School_Algorithm

a) جاگ جائیں

b) تیار ہو جائیں

c) لنچ باکس لیں

d) بس پکڑیں

e) بس سے اتریں

f) اسکول یا کالج میں داخل ہوں

15۔ کسی عدد کے فیکٹوریل کی تحسیب کے لیے ایک سوڈو کوڈ لکھیے۔ (اشارہ: 5 کے فیکٹوریل کو 5!=5 4 3 2 1 کی شکل میں لکھا جاتا ہے۔)

16۔ اس بات کی جانچ کرنے کے لیے کہ آیا دیا ہوا عدد آرم اسٹرانگ عدد ہے یا نہیں ایک فلوچارٹ بنائیے۔ تین ہندسی آرم اسٹرانگ ایک ایسا صحیح عدد ہے جس کے ہندسوں کے مکعب کا حاصل جمع اس عدد کے مساوی ہوتا ہے۔ مثال کے طور پر 371 ایک آرم اسٹرانگ عدد ہے کیوں کہ 3**3 + 7**3 + 1**3 = 371

17۔ ذیل میں ایک الگورتھم دیا گیا ہے جو اعداد کی درجہ بندی ایک ہندسی "Single Digit"، دو ہندسی "Double Digit" یا بڑا عدد "Big" کے طور پر کرتا ہے۔

Classify_Numbers_Algo

```
INPUT Number
IF Number < 9
     "Single Digit"
Else If Number < 99
     "Double Digit"
Else
     "Big"
```

اعداد (5, 9, 47, 99, 100 200) کے لیے اس الگورتھم کی تصدیق کیجیے اور اگر ضروری ہو تو اسے درست کیجیے۔

18۔ کچھ تحسیبات کے لیے ہم ایک ایسا الگورتھم چاہتے ہیں جو صرف 100 تک مثبت اعداد کو قبول کرتا ہے۔

نوٹ

Accept_1to100_Algo

```
INPUT Number
IF (0<= Number) AND (Number <= 100)
     ACCEPT
Else
     REJECT
```

a) یہ الگورتھم کن قدروں پر ناکام ہو جائے گا؟

b) کیا آپ اس الگورتھم میں ترمیم کر سکتے ہیں؟